A RAND NOTE

Evaluating Expert System Tools: A Framework and Methodology--Workshops

Jeff Rothenberg, Jody Paul, Iris Kameny, James R. Kipps, Marcy Swenson

July 1987

RAND

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under RAND's National Defense Research Institute, a federally funded research and development center supported by the Office of the Secretary of Defense and the Joint Chiefs of Staff, Contract No. MDA903-85-C-0030.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

A RAND NOTE

N-2603-DARPA

Evaluating Expert System Tools: A Framework and Methodology--Workshops

Jeff Rothenberg, Jody Paul, Iris Kameny,
James R. Kipps, Marcy Swenson

July 1987

Prepared for
The Defense Advanced Research Projects Agency

RAND

PREFACE

This Note describes two workshops held at The RAND Corporation in June and November 1986 in conjunction with a study conducted for the Information Science and Technology Office of the Defense Advanced Research Projects Agency (DARPA), under RAND's National Defense Research Institute (NDRI). The NDRI is a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. The study was undertaken to develop criteria for evaluating and selecting tools used to build expert systems. The Note should be of interest primarily to decisionmakers concerned with choosing such tools, i.e., managers of expert system development projects and developers of expert systems. It should also be of value to developers of expert system tools and artificial intelligence (AI) researchers investigating new expert system techniques.

The main results of the study are presented in companion RAND Report R-3542-DARPA, *Evaluating Expert System Tools: A Framework and Methodology*, by J. Rothenberg, J. Paul, I. Kameny, and J. Kipps, July 1987. This work draws heavily on the experience of expert system tool developers and users. The authors enhanced their own background in the field by studying and using a number of major tools, and by hosting two workshops: one for tool developers (representing seven commercial vendors) and one for tool users (representing over thirty expert system development projects). The workshops validated and refined the authors' ideas and provided both objective and anecdotal evidence about the state of current expert system tools and expert system research.

SUMMARY

Expert systems represent a new approach to solving problems with computers, using programs that explicitly embody human knowledge and expertise from a given problem domain. The expert system paradigm emphasizes the rapid generation of prototype systems whose behavior can be understood and refined by domain experts. Because the expert system approach differs from traditional software engineering, it has spawned a new class of tools that can provide considerable leverage in building expert systems. One of the first steps an expert system developer usually takes, therefore, is to survey the available tools and decide which, if any, is most appropriate to the task at hand. However, this evaluation is a complex task; its cost and the attendant risk of performing it ineffectively motivate the development of a rational, reliable strategy for evaluating expert system tools.

The authors have developed a framework of criteria for performing such evaluations, along with a methodology for tailoring and applying this framework to particular projects and problems. That work is described in companion RAND Report R-3542-DARPA, *Evaluating Expert System Tools: A Framework and Methodology*. To validate and refine the ideas presented in that study, and to obtain evidence, both objective and anecdotal, about the state of current expert system tools and expert system research and development in general, The RAND Corporation hosted two workshops in 1986, one for expert system tool developers and one for expert system tool users. This Note describes the two workshops and summarizes the discussions and conclusions presented there.

Participants in the tool developers' workshop generally agreed with the evaluation framework and the proposed criteria, but some felt that it is premature to attempt to develop evaluation benchmarks that can directly compare tools with one another. The workshop discussions led to a shifting and refinement of the evaluation contexts, resulting in additional criteria relevant to the fielding of expert systems (e.g., portability, integrability, and phased installation). Concerns not yet

being addressed by most tool developers (e.g., software engineering issues, novice tool user and interface issues) were identified, and it was recommended that RAND next host a workshop for expert system tool users, to capture their concerns and perspective.

Prospective participants for the tool users' workshop were asked to fill out two questionnaires. The first was used to screen and select appropriate attendees, and the second requested detailed information about the respondents' current work. The results of the users' workshop and the questionnaires showed that users generally agree that the tools are of significant value and that they provide a great advantage over building expert systems directly in a programming language (such.as LISP). Most users feel that current tools are well-designed, reasonably supported, and sufficiently powerful to justify their cost. The most frequently cited shortcomings of the tools are lack of speed and lack of explicit control over inferencing capabilities. Concern was also voiced about the tendency of vendors to release new versions that have not been rigorously debugged. As expert systems move from the conceptualizing and prototyping phases into development and fielding, there will be an increasingly urgent need for tool integration with information acquisition and distribution facilities such as database management systems (DBMSs), communication networks, and sensor input, as well as with other software and hardware environments and output devices.

CONTENTS

PREFACE iii

SUMMARY v

Section

I. TOOL DEVELOPERS' WORKSHOP 1

Overview 1

Summary of Results 3

Fielding and Development 7

Assessment Techniques 15

Novice Users 21

Other Results 27

II. TOOL USERS' WORKSHOP 31

Overview 31

Summary of Results 34

Concerns of New Tool Choosers and Users 60

Wish List 62

Appendix

A. LIST OF PARTICIPANTS 69

B. QUESTIONNAIRES 75

C. BENCHMARKS 93

D. QUESTIONNAIRE RESULTS 115

I. TOOL DEVELOPERS' WORKSHOP

OVERVIEW

On June 26-27, 1986, a workshop was held for expert system tool developers at The RAND Corporation in Santa Monica. The participants were technical representatives from seven expert system tool companies and RAND researchers (the participants are listed in App. A).

The purpose of the workshop was to make sure the RAND study team understood the perspective of the tool developers. Although we already had our initial framework and methodology fairly well defined, to avoid prejudicing the developers, we presented our ideas only after they had had a chance to present theirs. We requested that each attendee prepare a 20-minute presentation giving his own perspective on evaluation. To give the workshop some common structure, we suggested that these presentations focus on evaluation contexts (including some background on each developer's philosophy of product design, support, and customer relations) and propose high-level evaluation criteria. We also sent the developers a preliminary description of a sample "benchmark" problem, to give them time to prepare their own examples for the workshop.

The participants represented several of the most prominent commercial tool vendors plus a sampling of others. The attendees provided an excellent cross-section of tools, markets, and target environments. We took care to insure that the vendors understood what we were trying to get out of the workshop, so that they would send appropriate technical people who would leave their salesmanship at home. The results of this groundwork were highly gratifying: the attendees were all top-level design personnel with extensive experience in tool development and broad perspectives on both technical and commercial issues, and they came prepared to share ideas and solve problems. The results of the workshop are due in large part to the professionalism of the attendees and their willingness to work as open-minded system analysts rather than sales representatives.

Don Waterman of RAND opened the workshop with a reiteration of its goals. About half of the first day was then spent hearing and discussing the developers' presentations. This served to establish the points of view and concentrations of the various companies represented, as well as to establish a common terminological base for further interaction.

We then presented our initial tool evaluation framework, organized around contexts, tool capabilities, metrics, and "methods of evaluation" (which we now refer to as "assessment techniques"). We also addressed the use of benchmark problems (miniature applications) to test various aspects of a tool's representational capabilities and its use in development.

The participants then broke up into three working groups of four to five people each. Working Group 1 focused on issues involved in fielding or deploying expert systems. Working Group 2 addressed assessment techniques for applying metrics to expert system tools. Working Group 3 focused on the tool evaluation criteria that would be appropriate for a "novice" expert system developer (defined as someone with minimal experience in designing and building expert systems).

The focus and composition of these groups was determined by mutual interest and choice, moderated by some voluntary "load balancing." The remainder of the workshop alternated between working group sessions, feedback sessions to disseminate working group results, and individual or group "dumps," where ideas and results were permanently recorded in text files (all participants and groups were provided access to computer terminals).

In the rest of this section, we present a summary of the most important results, followed by detailed results of each of the working groups. We present the working group results as relatively "raw" data here to give a flavor of the richness and degree of overlap among the groups; these results are aggregated and analyzed in the summary and elsewhere in this note. Finally, we discuss other ideas and concepts that emerged in the developers' presentations and are not covered explicitly in the working group results.

SUMMARY OF RESULTS

The workshop was extremely rich in the exchange of ideas and concepts, as expressed in the results of the working groups. This section summarizes four areas in which the workshop contributed to our understanding:

1. Enhancement and validation of our evaluation dimensions and criteria.
2. Shifting and refinement of our evaluation contexts, resulting in additional criteria relevant to the fielding of expert systems.
3. Identification of concerns not yet being addressed by most tool developers (e.g., software engineering issues, novice tool user issues, and interface issues).
4. The recommendation that RAND host a future workshop for users of expert system tools, to capture their concerns and perspective.

Validation of Evaluation Dimensions and Criteria

For the most part, the tool developers agreed with our evaluation framework and our proposed criteria. The major results of the workshop in this area were:

- *Application characteristics* was added as an explicit evaluation dimension.
- The tool developers felt that it is premature to attempt to develop evaluation benchmarks that can directly compare tools with each other.
- Consensus was reached on a set of potential assessment techniques to be used for evaluation.

In our original focus on matching a tool to a problem, the characteristics of the problem were considered to be crucial to the evaluation process, yet extrinsic to the evaluation framework. The tool developers felt that the framework for evaluating a tool should include the characteristics of the application for which it will be used. As a result, we promoted application characteristics from an extrinsic factor to an actual dimension of our framework. As the reader will see in the next section, the tool users at our second workshop did not feel as strongly about this as the tool developers--perhaps because they were more interested in general-purpose tools. Also, the larger, more complete tools currently advertise support for hybrid reasoning and knowledge representation, and so are projecting themselves as general tools. While tool users may be reflecting this image, the tool developers may be more aware of the differences between their tools (or they may be searching for such differences in the interests of securing distinct market niches), and so may be more concerned with how these differences are related to application characteristics.

Many of the tool developers felt that it was still premature to expect to use any kind of benchmark to compare tools directly with each other. The reasons for this reluctance were:

- The results of applying small benchmarks will not scale to large applications, and so may be misleading.
- Many proposed criteria (e.g., ease of learning, ease of use) are difficult to benchmark.
- The expert system field is still immature and does not have well-established definitions. This makes tool characteristics and features difficult to compare across different tools, since similarly named features of different tools may be functionally quite different.
- If benchmarks are implemented by a different team for each tool, the differences among the results are as likely to reflect differences among the teams as differences among the tools. (This same problem exists in software engineering when trying to compare different languages or methodologies by having a benchmark problem implemented by different groups.)

Nevertheless, it was felt that standard benchmarks implemented and published by vendors would allow tool users to compare the vendors' preferred styles and best solutions to the given problems. We note that this approach would make it impossible to "cheat," since performance or cleverness bought at the price of clarity or elegance would be apparent in the published solutions. Furthermore, vendors could propose their own benchmarks to insure that important features of their tools were shown to good advantage; vendors would presumably implement those benchmarks that suited their tools, thereby providing users with comparisons of comparable capabilities of tools. Implementations of benchmarks would provide operational definitions of capabilities, thereby alleviating the problem of terminological confusion among similarly named features.

The tool developers agreed that a number of other assessment techniques are also potentially useful for evaluating expert system tools. The complete list included:

- Comparison of a tool to a standard
- Interviews
- Questionnaires
- Benchmark problems
- Case studies
- Library of expert system efforts
- Development of an expert system for tool evaluation

These techniques were further discussed and evaluated by tool users in our second workshop.

Criteria Relevant to Fielding Expert Systems

The tool developers were in agreement that the ability of a tool to support deployment of an end-product expert system was critically important and warranted additional emphasis in our evaluation scheme. Portability and the ease of integrating an expert system into the application (or "delivery") environment, given the hardware and software constraints imposed by that environment, were seen as particularly important. This problem is alleviated if the delivery environment is identical to the development environment. It may also be vital for tools to support phased installation of an expert system in environments where live data cannot be interrupted or compromised.

Identification of Concerns Not Yet Being Addressed

It was felt that a number of important issues are not yet being dealt with by many of the tools, due to the relative immaturity of the expert system field and the relatively small number of completed, delivered expert systems. The main areas of concern involved software engineering, the use of the tools by "novice" users with little experience in building expert systems, and user interface issues. The tool developers felt that as expert system technology emerges from its infancy and loses its innocence, it will become increasingly apparent that expert systems are "just" another type of software product, like a database management system (DBMS), that must operate in standard application, software, and hardware environments and integrate well with other software tools and capabilities.

The ultimate effectiveness of a fielded system rests squarely on its execution competence and performance. Reliability and maintainability loom as major software engineering issues yet to be faced by many of the existing tools: particularly important here are error handling and recovery and the ease of maintaining delivered software in the field. Multi-user application support (including concurrent data access, with the attendant database management issues) is also seen as a crucial element that is missing from many existing tools. Finally, the usability and acceptability of a fielded system depend heavily on its user interface: this implies that tools should provide support for designing and building powerful interfaces, as well as allowing a delivered system to be embedded in an existing interface in the application environment.

Additional tool evaluation criteria were defined that would be appropriate for a "novice" developing a prototype expert system. It was felt that one of the major difficulties facing novices is determining which tool features are required for their particular problem. The novice therefore needs assistance in classifying problems according to type (e.g., planning, diagnosis) and in determining which capabilities and features are needed for solving problems of a given type. (Note, however, that classifying problems is still an open research issue (Chandrasekaran, 1986).) Another difficulty facing novices is that of determining the size or complexity of a problem.

Of particular importance to the novice is the ease with which the tool can be learned and used. The documentation and training provided by the vendor must be of high quality and complete, and the tool must be easy to use. If domain knowledge must be represented in a LISP-like language, the novice may have difficulty learning the language and using the tool. Similarly, a novice may have trouble if using a tool requires dealing directly with the operating system and the language underlying the tool. Though novices may be domain experts, they may lack sufficient artificial intelligence (AI) expertise to solve certain problems appropriately using an expert system tool; in such cases, the availability of vendor support and consulting become important criteria.

Recommendation to Hold a Tool Users' Workshop

This recommendation by the tool developers led to our holding a tool users' workshop, described in Sec. II.

FIELDING AND DEVELOPMENT

Workshop participants in Working Group 1 considered the evaluation issues involved in fielding (i.e., deploying) expert systems. This included evaluating the deployment process itself (including integration and interoperability issues) and the execution effectiveness of the delivered system. A detailed description of the evaluation criteria discussed is given below under the categories:

- Fielding/deployment
- Multi-user support
- Reliability
- Execution
- Human interface
- Maintenance

Fielding/Deployment Criteria

The evaluation criteria relevant to fielding or deployment were grouped into portability issues, environment constraints (including hardware and software), transition, and the fielding process itself.

Portability. Given that many expert system development efforts have a 2- to 3-year deployment horizon, the eventual fielding hardware is likely to be unknowable at the outset of development. Only in a few cases can it be assumed that a target expert system will be deployed in the same environment that is used to develop it. An expert system tool must therefore support portable deployment. This requires language portability and interface portability (particularly graphics).

The language used to implement an expert system with a tool need not be the language in which the tool itself is implemented (though this is sometimes the case). Many tools define their own languages in which their users implement the bulk of their expert systems; but some tools also allow the user to "call out" into some underlying language, for example to provide access to the underlying system; this "call-out" language is typically the language in which the tool itself is implemented, but may be yet another language. Portability requires that the expert system implementation language and the "call-out" language (if any) be available in the delivery environment (or at least be cross-compilable to that environment). Failing this, a tool user must rewrite code in order to field an expert system after developing it. To further complicate matters, many tools provide some subset of their development facilities in the delivery environment (e.g., explanation tracing). In such cases, the tool itself must also be portable to provide these facilities as part of the fielded system. This requires the language in which the tool itself is implemented to be available in the delivery environment as well. The use of a standard programming language (such as C, Pascal, Ada,[1] or Common LISP) as an implementation language offers one solution to this problem.

[1]Ada is a registered trademark of the U.S. government (Ada Joint Program Office).

In addition to the language issue, there is the thorny problem of interface portability. Even among standard languages, interface standards are rare, and those that do exist are often low-level. To make use of windows, graphics, color, and input devices (such as mice), an expert system implementer generally has only two choices: either use the facilities provided by the tool itself (and hope that the tool vendor will solve the problem of porting these to the desired delivery environment), or implement special-purpose interface code targeted for the delivery environment (which may mean sacrificing the ability to try this interface out in the development environment, if these environments are incompatible).

Environment Constraints. Environment constraints include both hardware and software. Hardware considerations include the delivery cost of hardware per end-user. In some cases, a chosen market and target price may determine the hardware; in other cases, hardware requirements may determine the price and therefore the market. Hardware may also be determined by decree, e.g., for government-furnished equipment (GFE). The growth potential of the target system depends on the ability to expand the system over time through network environments and the ability to take advantage of larger primary and secondary memory capacities.

The operating system used by the tool (and under which the tool runs) may exert constraints on an expert system application in the areas of communication and networking, multi-tasking capability, and the use of virtual memory. The application environment may also impose constraints. Government applications may be subject to military specifications dealing with such things as TEMPEST compliance, hardware, security, and formal verification. Financial applications may impose auditing and accountability requirements along with a strong disposition toward traditional vendors such as IBM. Critical environments such as nuclear powerplant control may impose severe reliability and availability constraints.

The transition from the development system to the installed, delivered system may require interfacing to a network, to other software, and to live data. In many environments, this must not cause interruption of service or unavailability of data and may require phased delivery in which the expert system is phased in in parallel with existing procedures, so that no data is lost. Tool support for performing this transition painlessly and safely is therefore an important evaluation consideration.

The fielding process becomes easier if the tool can encapsulate hardware-specific aspects of the application (such as sensor input) and operating-system-specific aspects of the application (such as input/output (I/O) and communication capabilities). Fielding is certainly easier if the hardware, operating system, and implementation language(s) used for the fielded system are the same as those used for the development system, but this will not always be the case. Language issues also include the capability of the tool to encapsulate external calls by permitting applications to call programs or functions in another language and tool support to insure that the form of these calls will port to the delivery environment. If the hardware/software environments differ from development to delivery, then another important tool criterion is the ability to translate or cross-compile application code. This may include procedures within the tool itself and those in the "call-out" language. The evaluation concern is whether this is done by the tool, available as a service, or requires hand coding.

Issues in fielding from the development to the delivery environment also include portability of the development interface, such as whether both interfaces should be the same and, if not, what should be removed and what must be recoded (e.g., should debugging aspects be removed? Do graphics interfaces need to be recoded? Do dynamic rule generation capabilities need to be removed or disabled?). Another concern is with limitation of the application size due to the hardware and software constraints of the delivery environment.

A final issue is whether fielding can be done by a novice user. If the tool allows end-users to create expert systems, it is an advantage if the fielding process is sufficiently automated to allow those users to field their own expert systems with little or no help. Since expert system technology is typically aimed at application areas that are not highly formalized or well understood, expert systems are likely to require change and evolution even more often than traditional systems (for which life-cycle costs of enhancement and debugging are already often greater than 50 percent of total system cost). An expert system is likely to be under development throughout its entire fielded lifetime, requiring either "field refinement" or refinement in the development environment followed by refielding. If refinement requires cycling back to the development environment, its cost and difficulty will depend on how well the tool supports the fielding (or refielding) activity.

Multi-User Support Criteria

Many real-world applications require multi-user access, which in turn requires support for consistency and synchronization of concurrent access and contention handling for database (or knowledge-base) updates. Similarly, tools may need to support multiple views of knowledge bases to allow different users to see different parts or aspects of a knowledge base, and to provide different access privileges for different users.

These kinds of multi-user support may be required by two quite different sets of users in two different contexts: by multiple tool users during development of an expert system, or by multiple end-users interacting with the final expert system. In the former case, the emphasis is on configuration management, whereas in the latter case it is on knowledge-base access, but both cases may require concurrency, contention handling, views, and access privileges. These issues have not been addressed by most expert systems to date, and few of the available tools provide solutions to these problems.

Reliability Criteria

Reliability criteria are general software engineering issues that include availability, reliability and recovery, error reporting, validation and verification (V&V), and quality assurance. For all these issues, the relevant criteria for tool evaluation have to do with the support a tool provides, that is, how it helps a developer build an expert system that satisfies these requirements.

Availability (i.e., the percentage of time a system is available for use) and reliability are requirements of the application environment and apply to software, databases, and network interfaces as well as hardware. Recovery requires that a system maintain internal consistency in the face of hardware failures or inconsistent data or data access. Error handling requires that a system report errors to end-users in understandable terms; this must apply both to errors encountered by the application expert system and to errors encountered by the underlying tool in cases where the tool is present when running the application. V&V does not refer to formal verification but rather to insuring that "the right system gets built in the right way"; to support V&V, a tool must help an expert system developer build confidence in the design and implementation as it progresses. Support for quality assurance involves allowing a tool user to test and retest an expert system application easily, e.g., by making it easy to build test suites, keep failure statistics, run load tests, etc.

Execution Criteria

In the (highly nonstandardized) jargon of expert systems, "execution" usually refers to the execution of the target expert system. Execution criteria include performance, memory requirements, and integration capabilities. Performance is usually evaluated in terms of speed and real-time capability (where speed is a function of size and complexity). Speed criteria are applicable to rule execution, search space examination, interface interaction, external calls or communication, etc. Real-time capability is essentially the ability to guarantee speed requirements. It was noted that tools which implement expert systems in LISP dialects and perform traditional, synchronous garbage collection may have trouble meeting real-time requirements. Some newer LISPs, however, perform garbage collection in background, which should smooth out the associated delays; in addition, it would be possible to give LISP programmers explicit control over garbage collection, though this runs counter to the traditional LISP style of high-level programming.

Memory considerations include primary and secondary memory requirements and the need to optimize or reconfigure the memory of the fielded system by removing parts of the development environment (or paying the overhead of having unused or disabled tool features present in the fielded system). There are also memory tradeoffs to be considered between running in compiled or interpretive modes.

Integration capabilities include (1) encapsulating external interfaces to the operating system, other languages, databases, external applications, interprocess communication, asynchronous communication, and physical devices (i.e., defining them as logical devices); (2) architectural tool support for asynchronous communication (e.g., data-directed computation) and consistency maintenance (e.g., truth maintenance); and (3) "embedability"--allowing an application to be called as a function by other software or to be preempted by other processes, and generally allowing the delivered expert system to cooperate with existing software.

Human-Interface Criteria

Human-interface criteria apply both to a tool itself and to the delivered expert systems built with a tool. They include (1) "defeatability" (the ability to turn off selected features of the interface or of the application/tool environment); (2) interface consistency across various modules or modes within the tool or application itself and also between the tool or application and other preexisting interfaces in the user's environment (e.g., editors, operating system, etc.); (3) support for implementing help, documentation, and explanation; and (4) interface construction support (how well the tool supports construction of application system interfaces).

The first three of these are fairly self-explanatory. Interface construction support should include separability, modularity, embedability, mixed initiative, menus, commands, multi-tasking/windowing, and graphics. Separability and modularity allow an interface to be removed or changed independently of the rest of the supplied environment. Embedability is concerned with the ability of the target system to be merged into an existing target environment with an existing interface. Mixed initiative addresses the tool's support for building interfaces in which the user and the system share control of the interaction. Menu capability refers to the tool's support for building flexible menu interfaces. Command capability refers to the tool's support for building flexible command interfaces (i.e., providing facilities for parsing, spelling correction, command completion, etc.). Multi-tasking/windowing is concerned with the tool's ability to take advantage of existing multi-tasking windowing environments or to support the creation of such an environment for a target system. Graphics capability refers to the tool's support for providing graphic explanation facilities in the delivery environment and for providing other application-specific graphics.

Maintenance Criteria

Maintenance is another common software development concern. The nature of expert systems and the kinds of problems they typically attempt to solve give maintenance special significance, since these systems often require continual evolution and refinement, even after delivery. Relevant tool characteristics include (1) tool-supported configuration management, consistency maintenance (of application code, documentation, and explanation), and automated management of trouble reports and change requests; (2) leverage furnished by the tool for maintenance of target expert systems; (3) language issues (discussed previously); and (4) the maintenance process itself.

Maintenance support requires the ability to make the explanation and "introspection" facilities of an expert system evolve incrementally to show the internal state of the system (e.g., as a trace of rule firings), to help expert system developers and end-users understand the system's behavior. Capabilities to support explanation should allow different levels of user sophistication to address the different needs of developers and end-users. In addition, a tool should support modular code development, including techniques for "information hiding" and for building procedures and packages.

The maintenance process depends critically on whether maintenance changes require cycling back to the development environment and refielding or can be made directly in the delivery environment. The latter allows easier and quicker evolution or fixing of bugs but requires that facilities for testing, V&V, quality assurance, etc., be present in the delivery environment (which is rare). A related concern is whether maintenance of an expert system can be performed by end-users themselves (or by applications programmers in the end-user environment) or requires the services of the knowledge engineers who originally developed the system.

ASSESSMENT TECHNIQUES

Working Group 2 focused on assessment techniques (which at the time we called "evaluation methods") for evaluating expert system tools. A group report was produced as well as individual notes. A short abstract of each note is given below.

Abstracts of Individual Notes

Richard Fikes wrote on "Methodology for Evaluating the Functional Capability of Tool Components." In his note, he discusses functional capability benchmarks, gives an illustrative problem fragment, and describes how such benchmarks could be used for evaluation. He notes that certain characteristics such as debugging aids and the degree of difficulty in learning the tool cannot be tested with these kinds of benchmarks.

Steve Hardy addressed the issues in the development of expert systems by programmers and intended his note as a guide for developing criteria for evaluating expert system shells. He lists four areas of issues in expert system development and the dimensions over which they should be considered. He emphasizes the fact that in developing the evaluation criteria we must keep the intended user and purpose in mind, since almost all expert systems to date have been demonstration prototypes built by AI specialists.

Charles Riese's note included three topics: ideas about assessment techniques, a discussion of test cases for expert system tool evaluation, and a discussion of building an expert system to be used in the evaluation of expert systems.

Don Waterman's note describes different contexts for considering the tool evaluation problem and looks at important capabilities in each context and at assessment techniques for evaluating those capabilities.

Group Results

Our own work prior to the workshop had convinced us of the need for further thought on the subject of assessment techniques, i.e., ways of actually applying metrics to tools, and we had our own list of potential techniques, which we introduced into the working groups to stimulate discussion. Many of these same ideas were broached independently by other attendees, and several new ones were added. (For example, we considered the idea of a consultation expert system to advise a prospective tool user on how to evaluate tools throughout the project; this idea was also raised by several of the attendees at the workshop.) In what follows, we do not distinguish which ideas predated the workshop but simply present them all as "results."

The workshop produced seven potential assessment techniques that might be used in the evaluation of expert system tools:

- Comparing tools to a standard
- Conducting user interviews
- Asking users to fill out questionnaires
- Applying benchmark problems
- Performing case studies
- Gathering a library of expert system efforts
- Developing an expert system tool evaluation consultant

Comparison to a Standard. Using this technique, one would compare a capability of a tool with what some standard language (e.g., Common LISP) offers, or compare a feature of the tool with similar features of some ideal tool or of other tools (e.g., forward chaining in a given tool with forward chaining in ROSIE). This would produce a comparison of each feature (or capability) of the tool with the corresponding feature (or capability) of some baseline technology.

This technique presents several problems. The choice of a baseline is not an easy task in such a young technology. A baseline such as Common LISP may not be particularly useful for differentiating tools, since most or all tools would exceed its capabilities by a wide margin, which would nevertheless be difficult to measure with any precision. Comparison to an "ideal" tool presents the problem of defining this ideal.

Comparing tools with one another on the basis of tool characteristics and features would yield a sort of Consumer Report™ (deemed quite desirable by attendees of the second workshop); but such comparisons are difficult because characteristics may have the same name in different tools but be functionally quite different (this may lead to inadvertently comparing apples and oranges simply because they have both been named tangerines). A reviewer making such a comparison (and even moreso, a reader trying to interpret one) would need to have an in-depth understanding of all the tools being compared.

Interviews. Interviewing people who have completed long-term development efforts in order to learn from their experience with the tool(s) they have used is usually done informally by developers shopping for tools, but is generally limited to an arbitrarily chosen set of tool users (i.e., personal friends or other developers within the same organization). To be useful for evaluation, this technique would have to be applied systematically and objectively, and the results would have to be made widely available. It would also require continual updating to include newer and more relevant experiences.

User Questionnaires. This technique is analogous to interviewing tool users but is made more formal by the use of a standard questionnaire. Information collected from various users of a tool would be kept in an easily accessible (and continually updated) database. To be of maximum utility, questionnaires could be completed at several points during the development of an expert system; this would provide valuable information on long-term efforts and on how tools support the overall development process.

Benchmark Problems. Benchmarks are special problems developed to test the capabilities of an expert system tool. Problems would be stated in implementation-independent terms and could be solved by vendors offering expert system tools and/or by prospective tool users. Solutions for each tool would be published, along with such quantitative measurements as the time required to implement the solution, the resulting system size, etc. Solutions would be evaluated primarily on the basis of style and conceptual clarity, and only secondarily on the basis of their quantitative measurements. Specific criteria for evaluating solutions to given benchmarks would be developed iteratively in the literature, and solutions would attempt to optimize for these criteria as they evolve.

A small benchmark problem should be capable of solution in hours or days. It would consist of an informal statement of the desired capability, a specific problem fragment for testing the capability, and a description of the role the capability plays in solving this problem. Small benchmarks could be used to test such things as the expressive power of representation languages and the execution efficiency of various programming paradigms.

Benchmarks for testing functional capabilities could be obtained from tool vendors by asking them to describe what they feel are the most important capabilities offered by their tools. Each vendor would explain each capability in terms of the kinds of applications it serves and would provide a corresponding benchmark problem to test that capability, along with one or more solutions to that problem using the vendor's own tool. Each solution would be accompanied by a statement of the criteria that solution attempts to optimize, thereby minimizing the danger of judging a solution by inappropriate standards. The following is a skeletal example of a small benchmark problem:

> Experts may organize their domain knowledge around taxonomies: hierarchical structures in which properties of an entire class can be stated just once and then "inherited" by members of that class or any subclass.
>
> For example, all squares are rectangles and all rectangles are geometric figures. The area of any rectangle can be computed by multiplying its height by its width. The width of a square equals its height.
>
> Represent this knowledge and use it to compute the area of a square called "object-22," with a height of 5 meters. Rectangle height must be a number. Extend the representation so the system will reject nonnumeric heights for rectangles.

A large benchmark problem is one that can be solved in weeks or months. It would consist of a detailed description of the problem being addressed, a checklist of questions to which the implementer must respond during the development process (such as, "How hard was X to implement?" or "How long did it take to implement Y?"), and follow-up interviews to obtain subjective evaluations. An example of a large benchmark problem, that of locating and diagnosing a spill in a chemical plant, is presented in detail in Hayes-Roth, Waterman, and Lenat (1983).

As a result of the discussion of benchmarks, we solicited medium-scale benchmark problem statements from each of the vendors after the workshop. In order to provide a form and example, we first prepared our own sample benchmark (given in App. C) and distributed this to the attendees. In response to this request, we received a single vendor-generated benchmark proposal (from Radian Corporation), which is also reproduced in App. C.

Case Studies. A case study is a controlled recording of an expert system development effort that attempts to capture the relevant aspects of the tool being used. Unlike a benchmark, it focuses on a real development effort to solve a real problem, rather than dictating the problem to be solved, and is therefore not fully controlled. It seeks to instrument the development process in a real case, thereby avoiding the artificiality of an oversimplified problem. It would consist of a methodology for recording the history of an expert system development without imposing undue constraints on the developers (who would be expected to resist any overhead introduced by the instrumentation process).

A case study would be presented as a description of the problem being solved, along with a development history that would include both objective facts (e.g., overall development time, time to reach particular milestones, number of rules, size of knowledge base) and subjective factors (e.g., ease of use, naturalness of knowledge representation, problems encountered, successes obtained). There would also be post-development interviews. Case studies would be indexed by application type, domain, and various features of the development environment to allow prospective tool users to evaluate the applicability of a given study to their own problem and environment.

Library of Expert System Efforts. This would consist of a library of information about expert system development efforts, organized into a database that could be searched to find projects similar to a proposed project. Such a library would require indexing similar to that proposed for case studies above; it might, for example, match the "signature" of the proposed project (i.e., the characteristics of its application and development domains) against those of items in the library. The library could contain entries from all other categories of evaluation techniques: comparisons, interviews, questionnaires, benchmarks, and case studies, in addition to relevant literature, conference proceedings, technical papers, etc.

An Expert System for Tool Evaluation. A consultation and diagnostic expert system could be developed that would take a complete description of the proposed application as input from its user and produce as output issues to consider, questions to answer, and capabilities, features, and metrics that might be useful for evaluation. This expert system would apply rules based on tool criteria and problem attributes and access the library of expert system efforts as part of its knowledge base.

NOVICE USERS

Working Group 3 focused on criteria that would be relevant to choosing an expert system tool for use by "novices" developing a prototype expert system. In our context, a novice is someone acting as a knowledge engineer who has no AI expertise. This may be a domain expert (with or without any computer background) or a "vanilla" programmer, i.e., one who is competent in some general-purpose programming language, but is *not* an AI programmer. (These definitions of novice and vanilla programmer are not meant to be judgmental; they are merely convenient jargon for use in this Note.)

The group developed the following set of categories of evaluation criteria:

- Integration and embedding
- Application and domain types
- Problem scale
- Multiple/single user development
- Development environment
- Tool learnability
- Explanation
- Cost
- Future of the tool
- Support
- Techniques to help in tool selection
- Other issues

Integration and Embedding

These issues are more difficult to assess and address for a nonprogrammer domain expert than for a vanilla programmer (for whom there will be some carryover from building traditional systems). The issues involve accessing existing software packages (e.g., databases, spread sheets, statistical packages, graphics) for both input and output. This can be done either from within the tool or by accessing the tool from within an existing environment that provides access to these other packages. In some cases, it may be possible to simulate this integration within the tool itself, though this may not field well if the ultimate target environment is different from the development environment. The crucial criteria here are whether and to what extent a tool provides "hooks" into applications libraries or into other programming languages to allow implementing interfaces to other packages.

Application and Domain Types

Novices need help in understanding which applications and domain types may require special tool characteristics (such as support for multiple viewpoints, time sequences, histories, truth maintenance, real-time access to data, etc). There is currently a great deal of work going on in the research community to produce canonical characterizations of problem types, but consensus has not yet been achieved in this area. In the absence of standard characterizations, novices must at least be able to access examples of similar applications that have been developed by other novices (with similar backgrounds and experience) using the tools under consideration. A library of case studies and expert system efforts would probably be the most useful aid here.

Problem Scale

The novice has to be aware of many issues dealing with the scale of the problem. It is important to understand what problem sizes are reasonable for particular tools or, conversely, whether a given tool is adequate to support a given problem. Similarly, it is important to know whether a proposed application will perform at an acceptable level (with respect to speed and memory) on the target machine, using a given tool. If a problem requires building a large knowledge base, the tools to be considered must allow building and maintaining this knowledge base and must provide support for modularization (e.g., of code and rulesets). A related concern involves the ease with which a prototype solution can be scaled up to a fully developed system, and whether the same structure and approach will be appropriate for both. The safest course for a novice would probably be to consult case studies and vendor references to ascertain whether other novices have built similar systems of similar scale.

Multiple/Single-User Development

Novices working in a multi-user development environment may require special tool support for knowledge acquisition, identifying inconsistencies in a knowledge base, reporting system status, and building expert systems to support multiple end-users. The tool should make it easy to detect logical inconsistencies and inconsistent use of symbols in the knowledge base. Where the novice developers are the domain experts, this translates into a criterion of "knowledge acquisition" support; in any case, it requires knowledge-base configuration management to coordinate the activity of multiple developers. The definition of consistency in the knowledge base may also depend on whether the tool can support multiple results or conclusions: if so, novice tool users must be made aware that unintended inconsistencies may go undetected. Novices who are domain experts may also have heightened needs for knowledge-base browsing (e.g., to reconcile differences among multiple experts) and for the collection and maintenance of development history (particularly for nonprogrammers).

In addition to the above issues for multiple developers, if the target application is intended for multiple end-users, the tool may need to allow concurrency and to control contention; similarly, it may need to allow alternative views of the knowledge base and support knowledge-base merging. If the developers are nonprogrammers, their reliance on a tool's built-in support to solve software engineering problems such as these will be even greater.

Development Environment

The novice tool user's main concerns with the development environment will be with responsiveness and the functionality of the user interface. For novice developers serving in the role of domain experts, the task of "knowledge acquisition" (i.e., entering knowledge into the knowledge base) and knowledge representation may require specialized editing modes or support for various forms of knowledge entry (e.g., natural language, rules, examples, diagrammatic input, taxonomies, axioms, equations, assertions). Similarly, explanation, tracing, and debugging facilities (e.g., the ability to browse through a trace or to obtain English-like explanations) are crucial for novices, even seasoned programmers who are not familiar with the expert system approach. For nonprogrammers, a tool must also provide on-line help facilities, support for the graphics needed by the application, and a high-quality user interface which is robust, user-friendly, and ergonomic.

Novices are likely to be fairly intolerant of slow response. The development environment should interact quickly with the tool user when adding knowledge, editing, testing and rerunning with trivial changes, performing error detection and consistency checking, and loading and saving knowledge bases.

Tool Learnability

A novice must consider a number of issues related to learning to use a tool. The primary criterion here is the time it will take to learn to use the tool in order to become productive, proficient, or expert, or to be able to solve the problem at hand. To evaluate the available means to this end, the novice must consider the materials and support that are available from the tool vendor, e.g., tutorials (both documented and on-line), documentation, problem/program samples, training, etc. It is particularly important to determine whether these are intended only to teach the user to use the tool or to go beyond that and show how to solve problems.

Explanation

Explanation support must be evaluated with particular care, since in the absence of such support in a tool, building an explanation facility for the target expert system may be beyond the reach of most novices. It is useful for the tool to provide explanation in the development environment for knowledge-base tracing and debugging; but tools that provide such facilities do not always allow them to be exported easily into the target delivery environment. The novice must be careful to avoid such "dead ends" in the development process by looking ahead to fielding.

For an explanation facility to be useful to novices (both during development and after delivery), a tool must allow explaining any conclusion or action (along both *"HOW"* and *"WHY"* dimensions) in detail appropriate for the novice (e.g., presenting its explanations in English with helpful diagrammatic representations, and supporting the user in browsing through the explanation). In addition, it should be easy for the user to request and receive explanations at any time (e.g., during development, during consultation with the expert system, or after conclusions have been reached by the expert system).

Cost

Not surprisingly, the group felt that cost often functions as an initial filter of the tools to be considered. Cost considerations should begin by examining the startup costs of using a new technique or tool that is unfamiliar to a novice user. Given the difficulty of evaluating tools (especially for a novice who has little background on the subject), the cost of performing an evaluation to select an "optimal" expert system tool must be weighed against the tool's potential payoff. Furthermore, the novice must determine whether the cost of solving the problem using an expert system tool will yield a larger payoff (savings) than a more conventional solution. This requires evaluating both the expected cost of the tool (throughout its useful lifetime for the user) and the cost of the development effort (including maintenance) with and without the tool.

Future of the Tool

One very important aspect in selecting a tool is understanding the future growth directions the vendor intends for the tool and the future of the tool in the user's organization. The novice tool user must assess the future use of expert system tools in general within the organization and must consider which other problems the selected tool might be used for (including likely changes or extensions to the problem under consideration). A related issue is how well a tool fits into the existing computing environment. If the fit is not very good, the future directions of both the environment and the tools under consideration should be examined to see if there is likely to be a better fit with future versions of any of the tools.

The novice must also try to ascertain whether a chosen tool will allow scaling up a prototype problem solution into a deployable system appropriate to the expected delivery environment. This involves issues such as reliability, security, and hardware compatibility. It is necessary to evaluate the limitations of the tools being considered and to assess the future of those tools (i.e., does the vendor have plans to solve important limitations and when?). Finally, it is vital to make sure that knowledge bases can be preserved as a tool evolves.

Support

The novice tool user must carefully project the kinds of support that will be needed in the development and deployment phases and evaluate whether the vendor (or independent consultants) can supply the required support at an affordable cost. Support includes detailed documentation, advanced training courses, help with customizing the tool (or an expert system written with the tool), and help with porting applications to their target delivery environments.

Techniques to Help in Tool Selection

The novice tool user may utilize case studies to help select a tool. To be useful to novices, case studies of expert system developments should be independently conducted studies of real applications that have been developed by other novice users. They should include complete development histories (both subjective and objective), should be indexed by application type and domain over a wide range of problems, and should contain indications as to which features of the tool helped or hindered development. Case studies are most useful if they adhere to a standard methodology for recording and reporting results.

Other techniques suggested to help novices in tool selection include getting help from vendors, expert system courses, etc.; use of vendor demonstrations to show how a tool can be used by a novice and what can easily (and possibly) be done using the tool; contacting other users of a tool for references; and using a feature list as a guideline to understanding what features relate to solving the problem at hand and how easy it is to use those features (though this is subject to the caveats stated previously about comparing dissimilar features).

Other Issues

There remain several issues relevant to novice tool users that do not fall into the above categories. It is often claimed that one of the primary benefits of using expert system technology to solve a problem is that it leads to a better understanding of the problem (and the entire domain) by forcing knowledge to be structured and represented explicitly. In some cases, novices may consider this an important benefit of (and motivation for) using a tool, e.g., to train new domain experts. They may therefore evaluate a tool partly on the basis of how well its representation and methodology satisfy this criterion.

Novices should also be concerned with how well (or poorly) a tool hides the underlying language and environment. If solving the problem at hand will require escaping to the underlying language or environment, it is important to evaluate how difficult that will be.

Finally, the novice may be concerned with whether an understanding of AI techniques (e.g., certainty factors, truth maintenance, meta-knowledge) is required in order to use a particular tool to good advantage. In most cases, it will be an added burden for a novice to learn this extraneous subject matter, though in some cases, learning about AI may be one of the motivations for using an expert system tool.

OTHER RESULTS

This section presents additional ideas and concepts that emerged from the briefings given by the developers.

An interesting and controversial topic offered by Charles Riese (Radian Corporation) was that the evolution of an expert system should include the conversion of rule subsets into utility algorithms. An important tool characteristic would then be the support of an automatic path that could convert heuristic knowledge into algorithmic knowledge. Riese also expressed the view that when the expert system represents a small portion of the entire application, the selection of an expert system tool should have a correspondingly small influence on hardware and software decisions.

Steve Hardy (Teknowledge) expressed the view that tool builders need to treat expert system tools like other software systems (e.g, DBMSs) and develop a product philosophy that emphasizes good customer relations. This includes offering in-depth customer support, understanding the customer's business and how an expert system will fit into the environment and problem solution, becoming management consultants as well as software engineers, and offering extensive training. Proposed future tool characteristics include expressing the relationships between data in a nonprocedural way; being able to embed an expert system in a total system to DoD specifications (e.g., as an Ada implementation with no rotating storage on Mil Spec hardware); the ability to view a tool as a subroutine from conventional languages; and the ability to handle tools that are currently separate (such as expert systems, database management, and statistical packages) as libraries instead of separate tools.

Richard Fikes (Intellicorp) emphasized that knowledge acquisition is the bottleneck in the development of expert systems. What is needed is to let domain experts communicate with the tools in ways that are natural to them (e.g., taxonomies, logical assertions, structural models, diagrams, situation-specific decision rules) with a minimal need for them to reformulate their knowledge. Reasoning needs to be able to answer multiple types of questions and perform multiple types of tasks. To do so requires representing knowledge in a task-independent fashion.

Expert system solutions require using multiple tools that need to be well integrated with each other and that are capable of integration with other software (a point made by all the developers).

David Hornig (Carnegie Group) presented the philosophy that the use of an expert system tool should increase the productivity of application programmers and that applications written with the tool should run reasonably well. He expressed goals for the tool developers that included making tools easy for both experts and beginners to use, avoiding precipices (that is, being continuously extensible), and being capable of supporting large systems, while yielding small solutions to small problems.

Lowell Hawkinson (LISP Machines, Inc.) discussed the usability of tools, including allowing end-users to maintain knowledge bases, not requiring the end-user to know LISP or AI, providing user interfaces that combine natural language and graphics, representing knowledge in a way that is readable by the user, giving users control over consultation sessions, and providing high-level domain modeling tools and simulation facilities. Complete and powerful tools should include learning capabilities and hierarchical modeling of objects. Application to real-time problems requires the ability to work on many tasks concurrently, a built-in ability to handle time, and keeping histories of data and functions.

Mark Wright (Inference Corporation) discussed the need for data-directed reasoning (including distributed problem solving and monitoring) and for rule-based, modifiable programs, opportunistic problem solving, and dealing with unstructured problems. Representation capabilities should include reasoning about the search space, multi-level search spaces, and data-directed negation. Delivery system characteristics include the need for cheap machines, the ability to interface to existing systems (e.g., IBM OS written in assembler), C-based tools, and validation (which he acknowledged as an unsolved problem).

Anthony Magliero (Software Architecture & Engineering) discussed the importance of the operational data-processing setting, the ability to integrate existing information sources, the ability to operate on in-place equipment, the use of a prototype as an integral part of development (rather than as a throwaway), the need for verification and validation aids, the need to generate runtime packages to operate in many different environments, the need for multi-user support for a given body of data and knowledge, the use of abductive reasoning in explanation, the ability to develop systems based on high-level descriptions or requirements analysis generated by application specialists, and the ability to perform nonclassification reasoning by constructing solutions.

II. TOOL USERS' WORKSHOP

OVERVIEW

On November 3-4, 1986, we held a second workshop at The RAND Corporation for expert system tool users. Whereas the participants in the developers' workshop were technical representatives of commercial expert system tool vendors, participants in this users' workshop were selected on the basis of their experience in building expert systems and choosing expert system tools. Our intention was to bring together a representative cross-section of expert system tool users to evaluate, critique, and revise our tool evaluation framework and methodology.

This section summarizes the results of this two-day users' workshop. We first discuss the participants and summarize the events of the workshop. Then we discuss the results of the workshop, presenting both the conclusions drawn from a number of working groups and the general concerns voiced by participants.

Participants

We sent an announcement of the workshop to over 100 expert system tool users, and enclosed an initial questionnaire requesting information about themselves, their projects, and the expert system tools they had used. (This questionnaire is reproduced in App. B.) We were surprised by the overwhelming response and interest generated by this initial inquiry and were forced to turn away a number of users due to logistic limitations and our desire to keep the workshop small.

We selected 32 participants for the workshop, using the results of our initial questionnaire. We filtered potential participants on the basis of the tools they had used, the domains and tasks they had worked on, the scope and complexity of their projects, their experience level, and their affiliation. We sought a cross-section along all these axes, with a slight bias toward DoD contractors, based on the charter of our project. Although our 32 participants may not have been a truly representative sample of expert system tool users, we felt a group of this size would be adequate to provide a diversity of useful insights, while being small enough to keep the workshop manageable.

The attendees covered a wide range of experiences. Some had used commercially available tools (ranging from high-end, LISP-machine-based tools to low-end PC-based tools), while others had built their own "in-house" tools. They represented many domains, including the military, aerospace, finance, and manufacturing, and many different types of tasks, including fault diagnosis, planning, classification, design, and monitoring. They were involved in a wide range of projects that varied in terms of development team size, total level of effort (i.e., number of person-years), system size, and stage of development (e.g., prototyping, developing, fielding). Their experience levels also varied, though to insure some commonality of background, we invited only users with some programming experience (though not necessarily much AI experience). They represented both research organizations and commercial companies.

After winnowing the participants, we sent them a second, more comprehensive questionnaire (also reproduced in App. B). A list of participants is given in App. A, and the results of both questionnaires are discussed in App. D.

Workshop Activities

Jeff Rothenberg of RAND opened the workshop with a presentation of the project's charter and goals and a discussion of our evaluation framework and methodology. The results of our developers' workshop were also discussed, along with their impact on our framework. The presentation then summarized the results of the first questionnaires returned by the attendees, to give the participants an overview of who was present and what their experiences were.

The remainder of the first morning was devoted to the attendees' completing the second questionnaire (which not all of them had received prior to the workshop), as well as to having them meet each other and interact informally.

The bulk of the workshop was spent with the participants divided into four working groups of about equal size. The diversity of the attendees provided an extremely rich interaction and exchange which produced a great deal of useful insight. Each group included a member of the RAND project, who recorded what was said and kept the discussion from ranging too far afield.

During the first afternoon, these groups discussed various dimensions of our evaluation framework as well as other issues relevant to expert system tool evaluation. Each group was given copies of all the completed questionnaires and began by compiling the parts of the questionnaires pertaining to the dimensions they were discussing, so that these results could be used to guide the discussion. The groups broke up in the late afternoon, and a member of each group presented a summary of the group's results to the full workshop, as discussed in the summary of results below.

The second morning began with a briefing by Dr. John Marinuzzi from Los Alamos National Laboratory. He described an AI training facility and curriculum developed by the Los Alamos Knowledge Systems Laboratory in conjunction with Sandia National Laboratories to bring staff members up to speed in AI tools and techniques. He expressed his conviction that without such support, even an intelligent and highly motivated scientist is likely to fail in the attempt to become a proficient AI programmer, because the tools and techniques evolve faster than an unsupported individual can learn to use them. The Knowledge Systems Laboratory is an attempt to collect a critical mass of tools and expertise to be used in training technical staff members in expert system knowledge engineering.

During the rest of the second day, the participants were divided into groups according to their applications areas, i.e., aerospace, finance, military, and commercial applications. On this day, the group discussions were left open, providing the participants with a forum for voicing their own opinions and concerns about expert system development and the tools required for their applications. These discussions focused on such topics as integration of expert systems into existing environments, concerns of new tool users, definition of the "ideal" expert system tool, and methodologies for building expert systems. Late in the afternoon, the groups again presented summaries of their discussions, concluding the workshop.

SUMMARY OF RESULTS

We first present some general issues that the participants identified, along with those criteria that they felt had the greatest discriminating value in narrowing the choice of a tool. We then discuss the results in terms of the five dimensions of our framework, commenting on various aspects of each dimension.

General Issues

Evaluation Caveats. A number of participants brought up general issues about evaluation, listed below:

- Who is the evaluator?
- Who will measure and monitor the accuracy of an evaluation to detect biases?
- Who is the evaluation for?
- Will the evaluations be timely? How will they be kept up to date?

The preferred choice for an evaluator is a conscientious, impartial reviewer. In particular, evaluators should be free from biases and invulnerable to "political" pressures that can make selection a foregone conclusion and turn evaluation into a sham.

Concerns vary among different user groups, e.g., managers, technical staff, end-users. To be of value to a certain group, an evaluation should address that group's concerns without including superfluous information.

One recommendation (which our project had already taken) was to tie evaluation to general capabilities rather than to specific tools, so that evaluation results will not become obsolete too fast.

Discriminating Criteria. Another issue that was raised had to do with tool selection and ways of pruning the space of tools so that an evaluation can focus on a small set of tools. The criteria listed below were considered to be particularly effective for discriminating among commonly available tools, thus narrowing the set of tools to be considered:

- Cost
- Availability of tool on required hardware
- Integrability
- Range of applications

The tools to be considered may be quite different, depending on a project's software budget and other available resources (such as personnel and computation power). Only a limited set of tools may be available for a given hardware environment (e.g., LISP machines, mainframes, or PCs).

The need to integrate a tool (or an expert system built using a tool) with other software or hardware may sharply constrain the choice of tools. Choosing a tool to build a single simple application is quite different from choosing one that will be used for a wide range of applications.

Application Characteristics

The users' workshop resulted in significant expansion and fleshing-out of those aspects of the application characteristics dimension that deal with the problem for which a tool is being used. There are many differences, some subtle, in the requirements for different expert systems that may make a tool that is well suited for one not very effective for another. For example, desirable tool characteristics for a chemical analysis system may differ from those for monitoring a manufacturing plant. Similarly, a simulation task may have different requirements from a design task. By examining the problem to be solved, one can identify capabilities relevant to tool selection. The following application characteristics were discussed at the workshop:

- Problem domain
- Problem type
- Nature of domain knowledge
- Operational constraints
- Formal properties of the problem
- Problem size
- User-machine interaction
- Intended user community
- System autonomy
- Development team characteristics

Problem Domain. The problem domain is the area of knowledge to which the expert system will be applied. Participants agreed that grouping problems according to problem domain is helpful. For example, tools for a mathematical domain need the capability to do arithmetic processing, while CAD/CAM (Computer Aided Design/Computer Aided Manufacacturing) systems require good graphics facilities. These generalizations are useful in choosing the capabilities to concentrate on in tool selection, although they are not always relevant. The following list of problem domains is representative of those enumerated at the workshop:

- Aerospace
- Agriculture
- Business management
- CAD/CAM
- Chemistry
- Computer networking
- Earth sciences
- Electronics
- Engineering
- Finance (risk, loan analysis)
- Geology
- Information management
- Law
- Maintenance/repair
- Manufacturing
- Marketing/sales
- Mathematics
- Medicine
- Military science
- Physics
- Resource management
- Risk management
- Software engineering
- Space technology
- Telecommunications

Problem Type. Problem type refers to the generic category of knowledge engineering application (Hayes-Roth, Waterman, and Lenat, 1983) addressed by a particular expert system. Participants agreed that considering the kind of problem would benefit tool selection by helping to focus on specific capabilities. For example, an expert system for monitoring requires a tool with real-time reaction capability, whereas a simulation system requires a tool that provides temporal representation. The following problem types were developed at the workshop:

- Analysis
- Classification
- Conceptual modeling
- Control
- Data fusion
- Data tracking
- Debugging
- Design
- Diagnosis
- Forecasting
- Intelligent database access
- Interpretation
- Monitoring
- Planning
- Prediction
- Prescription
- Repair
- Resource allocation
- Risk management
- Scheduling
- Simulation

It is important to note, however, that few of the applications discussed at the workshop involved only a single problem type: most were a composite of subtasks involving several different problem types, making such characterization difficult. In addition, lists like the one above contain items at many different levels of abstraction (for example, debugging can be considered a special case of diagnosis, whereas simulation can involve nearly every other item on the list). For these reasons, the workshop attendees were skeptical about the chances of arriving at a meaningful list of problem types that are independent, primitive, and useful.

Nature of Domain Knowledge. The nature of domain knowledge was introduced at the workshop as another factor to consider in selecting an appropriate tool. For example, if the knowledge in the application domain is incomplete, unreliable, or uncertain, the tool may need to support uncertainty propagation or fuzzy logic. If experts are not available locally, a tool that can be brought to them may be more attractive than one that is tied to a stationary mainframe computer. Domain knowledge characteristics enumerated during the workshop are shown below:

- Knowledge source
  - Experts
  - Field data
  - Algorithms
  - Literature
- Expertise availability
  - Expense
  - Location
  - Willingness
- Agreement among experts
  - Consensus
  - Resolution of discrepancy
  - Sufficiency of expertise
- Stability of data/knowledge
  - Incomplete
  - Unreliable
  - Uncertain
  - Frequently updated
- Time sensitivity

Operational Constraints. The conditions under which an expert system is to work (its operational constraints) must also be considered during tool selection. A system which must run on battery power in the field has different support requirements from one that will operate in an air-conditioned office. Such constraints include:

- Execution speed
- System integration and compatibility
- Real-time operation
- Physical environment (controlled-climate, office, hostile)
- Hardware portability
- Verification/proof of correctness

The integration issue received special emphasis at the workshop and is dealt with in depth in a later section.

Formal Properties of the Problem. Formal properties of the problem account for the relationship between general problem characteristics and tool features that aid the construction of systems to attack such problems. A problem that has a strong algorithmic component in its solution may benefit from a standard programming-language approach rather than a heuristic one. Some tools are adept with numbers and formulas, while others have rich, expressive languages for representing objects and their relationships. The following is a representative selection of such properties:

- Problem decomposability
- Algorithmic/heuristic
- Symbolic/numeric

Problem Size. The problem-size component covers considerations of knowledge-base size and complexity. Key issues raised at the workshop are listed below:

- Domain size
- Extent of coverage
- Coverage depth
- Representation granularity
- Knowledge base organization
- Knowledge base access

Prime concerns are whether or not a given tool will handle the target knowledge base and whether adequate response times will be realizable.

User-Machine Interaction. The nature of user-machine interaction was perceived as a critical factor in expert system development and utility. Both the interaction between the system builder and the tool and between the end-user and the target expert system were considered. Desirable interface capabilities included graphics, sound, and mouse-entry. Emphasis was placed on how a finished expert system will appear to the end-user and (if the tool does not provide an adequate interface) how much effort the system builder must expend to enhance the interface.

Intended User Community. Potential users of an expert system are shown in the following list:

- Domain expert
- Computer-naive professional (with minimal computer experience)
- Office clerk
- Programmer
- AI expert (acting as a knowledge engineer)

Participants felt that the intended user community greatly influences the interface, especially the explanation facility and knowledge acquisition facility. The tool needs to provide proper levels of explanation for all potential end-users as well as for the expert system developers. The interface should be simple enough to use easily, and powerful enough so that it is not frustrating.

System Autonomy. The issue of system autonomy is concerned with the role of the fielded expert system. Two possibilities explored at the workshop were (1) the use of an expert system as a decision aid and (2) the creation of an autonomous expert system. Among the issues discussed were the need for graceful interaction of a decision aid with users, the need for a decision aid to be able to ask questions clearly, and the question of how to monitor the performance of an autonomous expert system.

Development Team Characteristics. There was a lack of consensus among the workshop participants about the importance and utility of development team characteristics for tool selection. One point of view held that the people involved in building an expert system are part of the overall environment and that their experiences, strengths, and weaknesses should be considered in choosing a tool. For example, if key personnel have experience with a particular tool, some of its shortcomings may be mitigated by the reduced overhead of not having to learn a new tool. Similarly, it may be essential to choose a tool that supports a computer-naive interface for knowledge acquisition from domain experts, depending on the makeup of the development team. Another viewpoint was that a development team is often brought together *after* a problem is understood rather than as a constraint or characteristic of the application, and that the team's characteristics should therefore not be used in tool selection.

Tool Capabilities

In general, the participants agreed with the capabilities and features we had enumerated prior to the workshop. However, they felt that an evaluation should consider capabilities only after narrowing down the set of candidate tools (i.e., after filtering by price, availability of required hardware, etc.).

Specific capabilities are discussed below with comments, criticisms and recommendations extracted during the workshop. This is not intended to be an exhaustive list. The first four capabilities are arranged in order of relative importance, while the others, about which the group had mixed feelings, are listed alphabetically.

Knowledge Acquisition. While considered too broad a topic to be captured by a single capability, knowledge acquisition was perceived by many to be something that is largely missing in current tools, yet is in the critical path of future expert system work. It was also suggested that if and when automated knowledge acquisition becomes a reality, it may be particularly difficult to measure its quality and effectiveness.

Explanation. This capability was ranked high, apparently because few of the participants felt it was adequately supported by any of the features of existing tools. There was a desire for multi-level explanation that varies according to end-user types (e.g., novice to expert), presentation styles (e.g., textual or graphic), scope and function (e.g., summary, detailed report, or tutorial), and audience types (i.e., expert system developer or end-user).

The group also felt it was desirable that explanations be more than just a summary of actions and inferences: that is, the reasons for those actions, ideally expressed in terms of a model of the domain, should also be available. Finally, there was a desire by some to have program access to the explanation facility, giving target systems better control over their explanations.

Internal Access. It was felt that a tool should at least provide an escape to its underlying implementation language. Ideally, it should also provide access to (and control of) various internal parameters. While there was concern that such a capability might degrade the integrity of the tool and that this might greatly complicate porting to the delivery environment, internal access was seen as critical to a tool's extensibility.

External Access. This capability was viewed as most important by those participants whose expert system projects had reached a relatively high degree of maturity and who were now confronted with the problem of integrating their systems into an existing computing environment. Subsidiary capabilities included communicating with external software, receipt of interrupts from external systems, and warnings for incompatible access requests.

Arithmetic Processing. This capability was viewed as necessary but not vital, and less important than handling knowledge. There was disagreement as to whether supporting features (e.g., arithmetic operators) should be embedded in the tool if they were already available through the external access capability.

Certainty Handling. There was some disagreement about exactly what this capability implies (e.g., ranges vs. discrete points) as well as about its usefulness. Some users felt that it was important for diagnosis but meaningless for planning, and some thought it was generally useless and misleading. There was also a question about why such a controversial capability should be built into a tool when, using internal access, a system developer should be able to implement any preferred style of certainty handling.

Concurrency. This capability was seen to have three aspects related to *performance*, *problem*, and *solution*, respectively. Performance aspects focus on features that use concurrency to improve system speed. Problem aspects focus on situations where a system must interact with multiple external "real-world" processes simultaneously. Solution aspects focus on features that enable a system to be written as interacting, autonomous subsystems. Although there are examples of tools that attempt to support some of these aspects (e.g., PICON™ supports problem concurrency), concurrency was generally viewed as a lacking but useful capability for expert system tools.

Consistency Checking. This capability, while desirable, introduced some questions, such as whether a tool's semantics are domain dependent or independent, whether actual system performance should be verified, and whether consistency checking should be performed as a static or dynamic process. There were also questions about what should happen when an inconsistency is found, i.e., whether the tool should automatically correct the error, warn the user, or abort computation.

Development Documentation. It was largely felt that this capability could be closely coupled to a tool's explanation facility. While automatic generation of development documentation was seen as a needed capability, especially for maintenance, there are questions about its granularity: should it document the entire system as a whole, its component subsystems, or individual concepts? It was pointed out that some current tools do not provide even the capability to manually add comments to rules.

Inference and Control. Several additional features were enumerated for this capability, such as event scheduling and message passing, as well as those control structures supported by conventional programming languages (e.g., iteration and subroutining). It was also recommended that those features supported by a tool should have clearly defined semantics (for example, specifying what type of conflict resolution is used).

Life Cycle. Although there was not a strong consensus, it was suggested that a tool's support for target system life cycle be included as a capability (i.e., tool support for the evolution of a target system from conception through delivery and maintenance). In addition, it was pointed out that the ability of a tool to transition from one context phase to another is extremely important and should be included explicitly under life-cycle considerations.

Optimization. There was some disagreement as to what this capability should imply, i.e., should it mean performance optimization? Space optimization? And what should happen if one impedes the other? A number of features were recommended for supporting this capability, including intelligent look-ahead, result caching, rule compilation, dynamically reordering rules, automatic rule modification, and the ability to port to fast delivery environments.

Metrics

Our original framework provided a lengthy set of metrics to measure the quality of particular aspects of an expert system tool. These are described in detail in our second questionnaire (see App. B), and are summarized below:

- Adequacy
- Availability
- Breadth
- Clarity
- Cognitive efficiency
- Coherence
- Flexibility
- Integration
- Maintainability
- Modularity
- Philosophy
- Portability

- Completeness
- Congruence
- Consistency
- Controllability
- Cost
- Defeatability
- Ease of use
- Efficiency
- Extensibility
- Power
- Reliability
- Responsiveness
- Robustness
- Scalability
- Sophistication
- Subsetability/separability
- Usability
- Learnability

We attempted to make the list exhaustive, but the participants felt that it was too long and that the meaning of many of the metrics overlapped. On the first day of the workshop, the group discussing this dimension chose to aggregate the metrics into six "higher-level" concepts that subsumed the original ones. Though there is still some overlap and ambiguity in these aggregated metrics, the group felt that they would be easier to work with, while capturing the same information.

Aggregated Metrics. The six aggregated metrics developed on the first day of the workshop are:

- Cost
- Flexibility
- Extensibility
- Clarity
- Efficiency
- Vendor Support

Cost. This includes not only the sales price of a tool, but also its hidden expenses, such as costs of training, integration, etc. Furthermore, it includes not only monetary cost, but also expenditures of resources such as time and effort.

Flexibility. This subsumes those metrics that deal with a tool's power and capabilities: representational power (i.e., basic data structures and reasoning mechanisms), adequacy to perform a given task or tasks, ease of use (i.e., both of the tool and of systems built with it), and sophistication. It was noted that flexibility may be antithetical to maintainability.

Extensibility. This deals with applying a tool in ways that are not directly supported or were unanticipated by the tool's developer. It includes breadth of applicability, access to system parameters, defeatability (the ease with which one can override a system parameter or function), ease of integration, portability, scalability, and subsetability.

Clarity. This subsumes those metrics that deal with relative ease or difficulty of understanding the basic operations of a tool: ease of use and usability (i.e., how much work is involved in doing something), cognitive efficiency (i.e., how many concepts must be kept in mind to use the tool), coherence of the tool's features, responsiveness (i.e., *how* the tool responds, rather than how fast), maintainability, modularity, and learnability.

Efficiency. This encompasses all aspects of a tool's responsiveness (i.e., *how fast* it responds) and its utilization of computational and memory resources. This metric also deals with the efficiency of the systems built with a tool.

Vendor Support. This consists of the quality of support supplied by the vendor and subsumes such metrics as vendor philosophy, system availability, reliability, portability, and robustness.

The Varying Importance of Metrics Over Phases. The group further examined how the relative importance of these metrics varies through the phases of an expert system project, i.e.,

- Exploration/conceptualization
- Prototyping/design
- Development/implementation
- Fielding/delivery
- Operation/maintenance

The relative importance of metrics across development phases (see Fig. 1) is suggestive of the qualitative relationships of importance for five of the six metrics. The cost metric does not appear on this graph because, while cost permeates all aspects of evaluation, it behaves uniquely: cost seems to be of most importance at the transitions from one phase to the next, where decisions are made to continue with a tool or switch to a different tool. Leaving cost aside, therefore, we discuss the behavior of the remaining five metrics in the graph.

Clarity is important throughout the entire life cycle. It starts high, stays high, and ends high, for different reasons during different phases. It is important to the beginning user learning the tool, who must be able to grasp its concepts and representations quickly. It is also important to the developer, who must be able to apply the tool's mechanisms effectively, refine an evolving knowledge base and build a target system that is easily comprehended and verified by domain experts (during development) and easily understood and used by end-users (after delivery). Finally, it is important to the maintainer of a target system, who must be able to understand and modify the existing knowledge base.

Flexibility, on the other hand, is of most importance during the initial stages of tool use, starting high and peaking somewhere during the prototyping phase. As choices for representation and control become fixed, flexibility decreases in importance, taking on an almost negative aspect as the need for maintainability rises.

Vendor support has relatively low importance at first, taking the form of training and coaching; but it rises rather quickly, staying high throughout the remainder of the life cycle, as the tool user moves out of the exploration phase and pushes the tool to its limits.

Extensibility is of minor importance until the tool user begins to fulfill specialized requirements of an application during design and implementation. Its importance drops during fielding (when system functionality has presumably stabilized), but it rises again during maintenance as the delivered system evolves or requires reintegration in an evolving target environment.

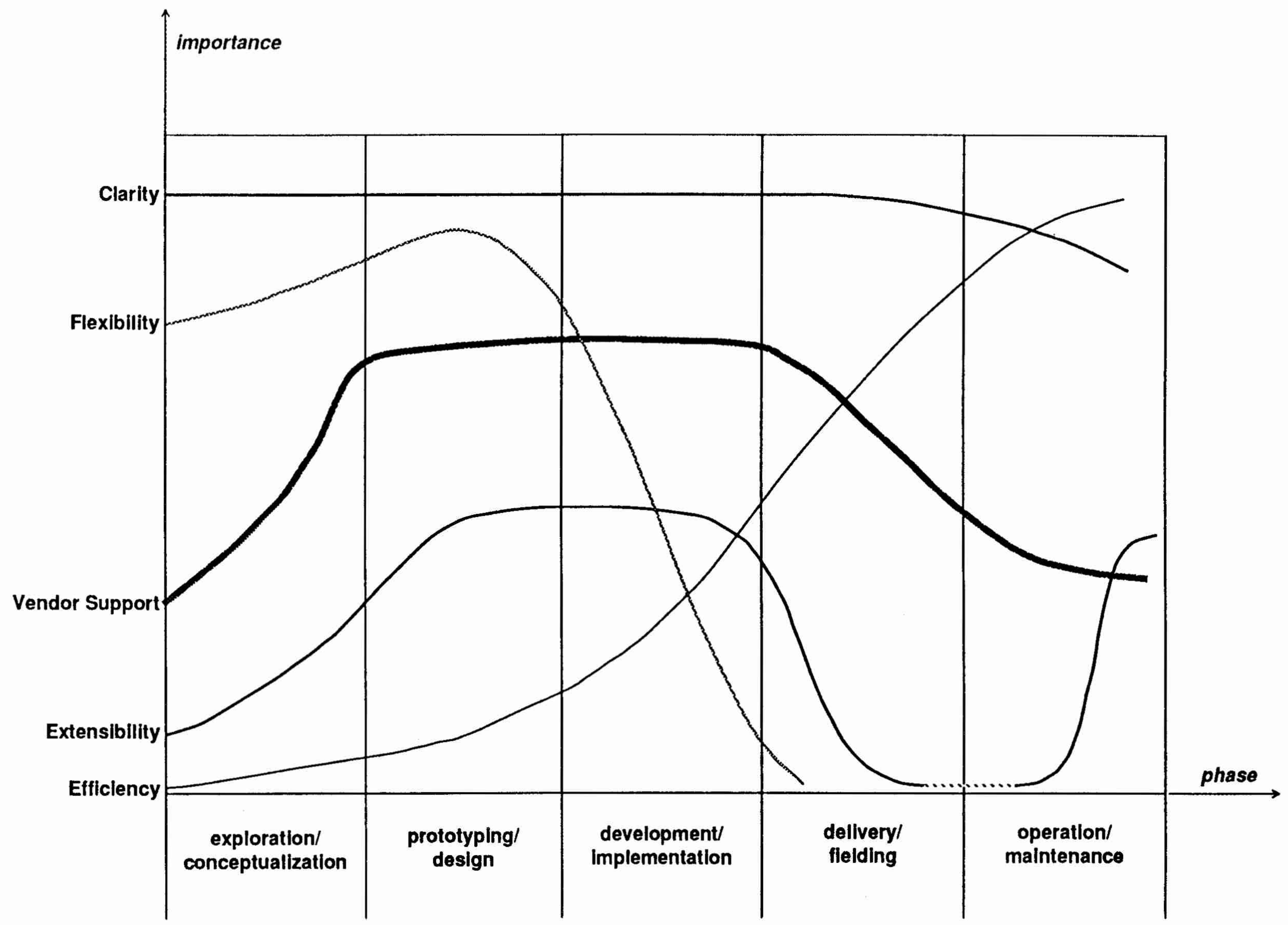


Fig. 1 -- Relative importance of metrics across development phases

Finally, efficiency remains low through exploration, design, and development, but ultimately becomes more important than even clarity if performance requirements become critical.

While this analysis of metrics is by no means definitive, it illustrates the concerns of the workshop participants that metrics be consolidated into a manageable set of concepts that are easily grasped. It also shows that the importance of these concepts varies through the life cycle of an expert system tool, making tool selection highly dependent upon the use (or uses) to which a tool will be put.

Assessment Techniques

Assessment techniques suggested in our original framework are shown below.

- Comparisons
  - between tools
  - between a tool and a baseline
  - between a tool and an ideal tool
- Benchmarks
  - small benchmarks
  - large benchmarks
- Case studies
- Library of expert system efforts
- Interviews
- Questionnaires
- An expert system for expert system evaluation

The working group that dealt with assessment techniques questioned a number of these, refined others, and added a new one (which it dubbed the "Rolodex"™ approach).

While all the techniques were considered to be of some value, the group doubted that the anticipated benefits of comparison with a baseline or ideal tool standard and large benchmarks would outweigh the costs of implementing these approaches (however, see the comparison between tools, below, for further justification of these techniques).

The techniques are discussed next in order of the importance accorded them by the workshop participants.

Comparison Between Tools. The users felt that a "Consumer Report"™ style comparison between tools would be the most beneficial assessment technique, since evaluation is generally motivated by the need to make a selection, for which a comparison of choices is particularly useful.

The point was raised, however, that there is no standard definition for many of the features and capabilities found in expert system tools. Two tools may claim to provide a forward-chaining capability, but their definitions (and implementations) of this capability may vary widely. In addition, a vendor may implement a crude form of some capability simply to be able to say that it is supported, even if it is not integrated into the system. To combat this, the comparison must supply a set of standard definitions and discuss the ways in which the features and capabilities of an expert system tool differ from this standard. For example, "goal-directed reasoning" might be defined as a capability for deriving a series of actions sufficient to achieve a stated goal; "backward-chaining" might then be defined as a feature that allows rules to be used to perform goal-directed reasoning. In this way, the report could indicate to what extent a tool has a certain capability and how it compares to a similar capability of another tool.

Comparisons between a baseline tool and an ideal tool were both felt to be impractical and unrevealing. For instance, most tools would so far exceed a baseline such as Common LISP that comparison would not be meaningful, while defining a standard, "ideal" tool that would keep up with advances in technology and be acceptable to everyone would present a major problem. However, we note that the realistic implementation of a Consumer Report comparison, as discussed above, requires standard definitions of capabilities, which may be equivalent to defining a baseline or ideal tool.

Small Benchmarks. A suite of small benchmarks was perceived as being helpful in testing a tool's capabilities, though there were some questions about how benchmarks could be implemented effectively. In particular, concerns about benchmarks included:

1. Each benchmark should test a certain capability and demonstrate how that capability integrates with the system as a whole.
2. A benchmark problem should, at least at an intuitive level, scale up to larger problems.
3. There should be benchmarks to test standard software engineering needs, including integration, reliability, and efficiency.

Despite the potential value of benchmarks, there were additional concerns over who would define benchmark problems and who would solve them. Obviously, the effort would need to be monitored by a group that was unbiased and conscientious. One suggestion was to have the tool vendors themselves define benchmark problems. Another suggestion was that the small "system teasers" often published in various trade journals could serve as an existing source of benchmarks.

Since each benchmark problem would have to be solved for every tool being evaluated, the number of solutions would tax an individual evaluator. Further, unless the implementers were already AI experts, their own performance might improve over time, producing better and better solutions with each successive tool. Alternatively, while allowing the tool developers to solve the benchmarks might deliver the most elegant solutions for each, it would not guarantee the most straightforward or revealing solutions. Finally, allowing different people to solve different benchmarks for different tools is problematic because variations in programming ability might introduce more variance than the tool characteristics themselves. It was felt that the best solution for this dilemma might be to use a combination of these approaches and compare their results. (For further discussion of benchmarks, see Validation of Evaluation Dimensions and Criteria, pp. 3-5.)

The majority of participants felt that large benchmarks would tend to be domain-specific. That is, the domain-dependent details of a large benchmark might make its relevance for different problems difficult to see. Furthermore, it was felt that if small benchmarks were scalable, large benchmarks might be superfluous.

A Knowledge-Based System for Tool Evaluation. It was felt that an expert system for aiding at least in preliminary evaluations of expert system tools would be helpful as a means of filtering through large amounts of initial data. Such a system could be used for either of two tasks:

1. Given a problem description, the system would identify those metrics that would be important for tool evaluation.
2. Given the metrics that are most important to the user, the system would recommend a tool.

Task 1 was generally considered to be infeasible at this time, since it would involve describing a problem to an expert system. However, Task 2 was considered tractable. Two approaches to building such a system were suggested: either as an automated tool for compiling data collected by other assessment techniques, or as a standard expert system application, drawing on the expertise of tool users with experience in selecting tools.

The first of these approaches to Task 2 was considered feasible, though it was noted that it depends heavily on the maturity of the other assessment techniques. The second approach was considered intractable for reasons similar to those that led to the rejection of Task 1: many users felt that the decision processes involved were so complex that this would again amount to describing the problem to the system, and that current "expert" users (including themselves) were not proficient enough at tool selection to be considered experts.

As an aside, one of the working groups on the second day of the workshop developed some evaluation scenarios as an exercise, feeling that if the group members could agree in most cases, this would imply that they were all using a common set of rules about evaluation. (Even if these rules could not yet be articulated, their implied existence would suggest the feasibility of knowledge-based tool evaluation.) The group did tend to agree on the evaluation process and results, which suggests that there is a pool of expertise that could be encoded. On the other hand, one participant related that students at an in-house AI training school had tried a similar task as an exercise and were unsuccessful, suggesting that this approach be pursued with caution.

The Rolodex™ Technique. A new approach to tool selection and evaluation that was suggested and received considerable interest at the workshop was one that was dubbed the "Rolodex" technique. The essence of this technique is that users should consult their personal or professional address books (files of business cards, etc.) and talk with someone who has used the tool.

The advantage of this approach is that people normally tend to interpret written recommendations with considerable skepticism (even those written by someone they know) and often prefer direct human interaction. It was asserted that this is the way consumers tend to buy first-time purchases (for example, consumers will call a friend and ask, "What kind of VCR do you have? Do you like it? Why?"). This gives them direct, personal input from someone they know and trust as well as the ability to interact and ask specific questions. The group felt that many users selecting an expert system tool do the same thing: they call business associates (or people they have met at conferences, etc.) and ask about their experiences with such tools.

Although concerns for privacy might prevent this from becoming a formal assessment technique, it should be acknowledged as a technique that people are likely to use in conjunction with other techniques.

Case Studies. While there was not a great deal of enthusiasm among the workshop attendees for using case studies as an assessment technique for evaluation, there was some discussion of possible kinds of case studies. In particular, it was noted that case studies of the tool selection process itself might be useful, as well as case studies of expert system development efforts that use tools.

However, there was general concern that such studies would quickly become outdated, and that they might produce too much data for practical decisionmaking. Because after-the-fact questionnaires tend to overlook errors and problems encountered during a project, the recommended approach to performing case studies was to make them external and progressive. That is, a development team would be visited at intervals and asked questions that would (among other things) attempt to identify progress and pitfalls. It was noted that one problem with this technique is that many of the most interesting projects are secret or proprietary, and hence cannot be studied externally.

Other Techniques. The attendees felt that the other techniques (i.e., a library of system efforts, interviews, and questionnaires) were so problem-specific that they would not be worth the time required to develop and implement them. The general consensus was that such time and money would be better spent on the other techniques. We note, however, that the results of the two questionnaires we sent to the attendees revealed some interesting insights which were purchased at the relatively low price of designing, mailing, and reviewing these textual instruments, without the need to purchase hardware or software or to write code.

Contexts

For purposes of evaluation, the context dimension accounts for the phase or phases of system development for which the tool will be used. Ideally, a single tool would be usable in all contexts, but a different tool might be used at each phase. We note that our breakdown of phases is somewhat different from the standard decomposition of the phases of expert system development (Waterman, 1986a).

Participants in our users' workshop elaborated this dimension, unifying it with the stages of expert system development:

- Conceptualization
- Prototyping
- Development
- Operation/fielding

Conceptualization. The conceptualization context encompasses the identification, conceptualization, and formalization phases of expert system development. A tool may be used as a structured formalism to aid in the design of the expert system and to support the development team in becoming familiar with the domain. The tool can help in decomposing the problem, identifying and organizing key concepts, and identifying the scope of the problem.

Prototyping. The prototype context is concerned with the use of a tool in prototyping an expert system. This context emphasizes the tool's facilities for guiding rapid development, eliciting different approaches and representations, and quickly trying alternative implementations.

Development. This context considers the tool as it is used to develop an expert system targeted ultimately for fielding. The tool's suitability for software development, including its debugging facilities and configuration control, are emphasized here.

Operation/Fielding. The operation/fielding context recognizes the effect that a tool has on the delivery of an expert system to its community of end-users and the performance and interface capabilities of that system. The emphasis in this context is on the tool's facilities for porting from the development environment to the delivery environment and its performance, maintenance, and support characteristics in the delivery environment.

The working groups agreed that in addition to being useful at each of these phases of development, a tool must also ease the transition from one context to the next. For instance, a tool that allows its debugging features to be turned off and its interface to be easily enhanced aids the developer's task when the expert system makes the transition from development to operation/fielding. The extent to which a tool supports these transitions was felt to be important by all participants.

Integration

A great deal of interest and concern was expressed at the workshop over issues involved with integrating expert systems with other software and hardware. Expert systems have evolved from special-purpose, standalone systems that accommodate AI expert users into multi-user systems that need to interact with on-line databases, be embedded in other programs, receive information from sensors, and use their results to control other hardware.

Many workshop participants had run into problems getting their tools to integrate with other computer systems or databases, and all agreed that substantial improvements are necessary. A few were designing extremely large expert systems that needed to handle billions of transactions per day. These developers encountered several integration problems, including accessing very large databases, integrating into a system that supports hundreds or thousands of concurrent users at remote terminals, and interfacing with huge mainframe computers. They were frustrated by the difficulty of accomplishing these tasks, due partly to the fact that the tools were not designed with integration in mind and the internals of the tools were not easily accessible.

Tool Support for Integration. Users discussed how much responsibility tool vendors should have for making their tools integrable. Several alternatives were discussed at the workshop:

- Standard applications
- Standard interface
- Internal access
- Interface management tools

Standard Applications. In the standard-application approach, a vendor chooses some popular standard systems (such as dBase-II™ of Ashton-Tate, Inc., or Britton-Lee Intelligent Database Machine™ and provides interfaces to them. This may satisfy some users for a limited time, but there will always be other software or hardware that will resist integration with the tool. The choice of a tool may therefore be dictated by which tools offer the required interfaces.

Standard Interface. The standard-interface approach requires agreement among vendors to standardize their interfaces, similar to the open-system architecture. The attendees felt that this was a promising alternative, but a distant goal due to the difficulty of getting large groups to form a consensus. Nevertheless, participants felt that the success of the ISO standard-communication protocols was an encouraging example, and that this approach should be pursued.

Internal Access. For the internal-access approach, the vendors would explain how to access their tools' internals so that users could write their own interfaces. This was also perceived as a positive step in the right direction. At least it would provide tool users with a way to interface with external hardware and software. Interfaces written by users might be included in future versions of the tools or put in the public domain. Of course, in cases where vendors do *not* take over such interfaces, users must support and maintain them themselves; this may require continual revision of these locally developed interfaces to adapt to new vendor software releases.

Interface Management Tools. Finally, the vendors could offer tools to users to help write their own interfaces, i.e., an interface-management-tools approach. This was appealing to most users and was perceived as having great potential value. Such tools would allow users to write whatever interfaces they needed, without grappling with the details of the tool's implementation.

Kinds of Integration. Integration is perceived as an urgent need, independent of which alternative is chosen by a vendor. Participants felt that the next generation of tools should address a number of integration issues, a selection of which are shown in the following list. A brief synopsis and examples of each were developed at the workshop.

- Information acquisition and distribution
  - Database management systems
  - Communications
  - Data input
  - Multi-user
- Environments
  - Software systems
  - Hardware systems
  - Output devices
- Temporal
  - Distributed
  - Concurrent
  - Real-time

Information Acquisition and Distribution

Database Management Systems. Many applications require that the expert system access knowledge from outside sources, because the knowledge must be updated, changed, or shared. If an external database already exists, it may be inappropriate to reproduce the data as a local knowledge base; translating an external database into a local knowledge base may even be impossible due to storage limitations of the expert system tool. Similarly, it may be impractical to include certain kinds of externally represented information, such as maps or weather data, in an expert system's knowledge base.

Communications. The expert system may need to communicate with other computers over communication networks, perhaps to update the knowledge base. System results may be sent over a modem to a remote location. For example, an expert system that plans flight paths for airplanes may send the airplanes their orders directly.

Data Input. Expert systems should be able to take advantage of sensors and other data-collection devices, such as vision systems, and exploit the edge they provide over entering information by hand or indirectly. For example, an expert system at a bank might be able to read information from bank checks.

Multi-User. This may be especially important if an expert system is linked to a central database. For example, a credit card approval expert system has to deal with stores dialing in for approval, new credit account entries, and updates to the database.

Environments

Software Systems. It is often necessary to link an expert system with other software. The expert system's results may be used by another program, or some other program may trigger the execution of an expert system. For instance, an expert system to monitor a spacecraft must interface with the spacecraft's other control systems.

Hardware Systems. Fielding a mass-distributed expert system on development hardware may be prohibitively expensive. Expensive equipment may be necessary for expert system design and prototyping, e.g., to provide graphic knowledge-base tracing and debugging, but the target expert system may need to be ported to a small computer or one that is more mobile (e.g., for use in an airplane) and may not require graphics or other support features.

Output Devices. The effectiveness of an expert system depends greatly on the nature of its communication with the end-user. It may exploit a graphical display to convey ideas more quickly than text by using windows, graphics devices, or printers.

Temporal

Distributed. A single expert system may not have the power to attack a whole problem. In this case, the problem can be split among several systems which communicate and cooperate with one another. For example, an aircraft design problem might be partitioned into the design of the fuselage, the engines, and the interior. Each system would need to communicate with the others to assure that all pieces would fit together and maintain conceptual homogeneity.

Concurrent. For some problems, no single machine has the computing power needed to provide a reasonable response time. Many alternatives may be explored simultaneously to improve performance. For instance, an expert system trying to devise a battle plan may test several prospective plans at the same time, selecting the most successful one when a choice becomes necessary.

Real-Time. Certain kinds of expertise must be applied in real-time, either due to a crisis situation (such as in an intensive care unit monitoring system) or because a slower response to a large quantity of queries would result in an unacceptable backlog (as with a passenger reservation system).

CONCERNS OF NEW TOOL CHOOSERS AND USERS

Another area of particular interest at the users' workshop was the situation faced by users selecting or using a tool for the first time. Many companies are just beginning to become involved with expert system technology. A user faced with selecting a tool for the first time has limited experience to draw on; a user who is receiving an initial introduction to using an expert system tool has needs that differ, sometimes substantially, from those of users with more experience. The tools must address these needs as well.

Workshop attendees who were at the stage of initial tool selection felt that it would be very helpful to read a consolidation of different opinions of people who had used the tools. These users felt that any information they could find would be worth spending the time to read. In contrast, those who had substantial experience with expert systems felt that such information would be of limited utility and perhaps not worth the investment in time. A useful analogy might be someone trying to sail a boat for the first time: the novice would want calm conditions, a working boat, and the best possible teacher. These types of requirements apply to a first-time expert system tool user too. Participants felt it was important to bring these issues to light even though they are primarily common sense. The following two contrasting anecdotes from the workshop point out key considerations about tool users' needs and the differences between those of new and experienced users:

One user had just bought a tool and had the misfortune of buying it when a new release had just come out. He had spent quite a while trying to learn to use the tool and was frustrated because he was never sure whether the problems he encountered were due to his own confusion, or whether the new version of the tool was still "buggy." He saw the vendor's dilemma too: if the vendor had offered him only the old version, he might not have bought it, since some of the capabilities required by his project were available only in the new version. But since the vendor had sold him the new version, he saw the vendor as unreliable, and he experienced frustration using the tool.

Another user had been using a tool for a while and had received the new version of that tool. He had some problems with the new tool, so he called the vendor. The vendor was immediately available to talk to him on the phone, and together they determined that there was a bug in the new version of the tool. The user had a new, patched version within two days. He was very satisfied with the vendor and did not mind having to help debug the tool as long as he got such quick response and support.

There was general agreement on a set of things that every new user needs:

- Working version (minimal bugs)
- Training course from vendor
- Immediate reinforcement

The workshop participants felt it was imperative that the version of the tool used for learning be one without bugs. Someone just learning to use a software package cannot distinguish between his own errors and possible bugs in the software. Sometimes only the very latest and therefore somewhat error-prone version has the features needed to write a particular application, but even in this situation it was deemed better to let the user learn about the tool with an earlier, insufficient version and move to the later, less stable version once the earlier one has been mastered.

If taking a course to learn a tool provides a significant advantage over reading the documentation, such courses should be taken whenever possible. The company involved in the design of a tool has significant insight about how that tool can best be used, and the chances of conveying that insight are much better in person. In addition, there is little chance of users retaining what they have learned in a course if they are immediately sidetracked onto some unrelated project. Workshop attendees agreed that if the investment in attending a course has been made, users should be allowed to set aside several weeks after their return to consolidate and reinforce what they have learned.

WISH LIST

In the two-day discussion of tools during the users' workshop, many participants were overheard saying, "Don't you wish there were a tool that did X?" Several of the working groups produced their own "wish lists" of desirable features. Similarly, the questionnaires revealed many desiderata for expert system tools. This section is a compilation of these wish lists, reflecting the needs and wants of a representative sample of tool users. (This is a somewhat random sample of ideas and does not necessarily represent our own desires or predictions for the next generation of tools, but we include it for completeness.)

The wishes fell into the following categories:

- Knowledge-base and representational enhancements
- Software engineering aids
- Delivery support

Knowledge-Base and Representational Enhancements

Workshop participants identified the need for a number of extensions in the areas of knowledge representation and knowledge-base maintenance:

- Higher-level knowledge representation
- Multiple relations
- Standardized knowledge representation
- Models of external entities
- Knowledge-acquisition aids
- Reusable general knowledge bases
- Self-organizing knowledge bases with automatic summary
- Automated validation and verification
- Retention of test-case data
- Change logs

Higher-Level Knowledge Representation. Representations such as rules and frames are sometimes too low-level for very complicated knowledge bases. A higher-level language written on top of these constructs would allow high-level knowledge representation. This makes it easier for experts to understand the knowledge encoded in the knowledge base and to find inconsistencies or gaps. Similarly, domain-specific input and output would help experts and developers understand and change the knowledge by presenting it in a familiar form.

Multiple Relations. Complex expert systems often require multiple relationships among objects, such as *IS-A (type-of)*, *part-of*, *subset-of*, *is-connected-to*, etc. Each of these requires a different kind of associated inference, analogous to "inheritance." (Such inference mechanisms are often referred to as different kinds of inheritance, which can be misleading, since inheritance per se pertains to the *IS-A* relation.) For example, the *part-of* relation requires inference for combining parts into wholes and disaggregating wholes into their parts. Tools should support these different kinds of inference or facilitate users' implementing their own.

Standardized Knowledge Representation. Many expert system applications deal with the same domains. A standard language for knowledge representation would allow information to be shared among applications. Similarly, communicating expert systems require a common language and protocol; this can currently be accomplished only by using a common tool (or programming language) for both applications. A standard would allow technology and information to be shared more easily. The requirements for a standard language include clarity, conceptual simplicity, and a high level of abstraction.

Models of External Entities. An expert system used in designing something may be concerned with only a part of the design, but it may still need to know about the other pieces (e.g., their weights, capacities, functions, etc). That is, it will require a model of each external entity in order to design the part. There is frequently a need to represent such external entities within an expert system, and tools should provide facilities for this kind of modeling.

Knowledge-Acquisition Aids. Knowledge acquisition is a major part of the development of any expert system. If this task could be simplified, clarified, or aided in any way, it would speed development and produce better systems. One suggested approach would be for tools to provide domain-specific knowledge acquisition (e.g., geared to physics or medicine) or problem-type-specific knowledge acquisition (e.g., for acquiring planning or diagnostic strategies). It would also be useful for a knowledge-acquisition mechanism to be able to draw conclusions from examples or to learn from its own mistakes. This would be analogous to field training of human experts. Knowledge-base browsers would also help developers organize or add to knowledge bases.

Reusable General Knowledge Bases. Ultimately, it is desirable to produce reusable knowledge bases for particular domains. For example, a general biology knowledge base might be useful across a wide range of applications. This could be thought of as an AI representation of the knowledge in a biology textbook; it would save developers the effort of encoding well-known background knowledge of biology, both for biological applications and for nonbiological applications that require this background knowledge.

Self-Organizing Knowledge Bases with Automatic Summary. Users expressed a desire for tools that would automatically organize their own knowledge bases, i.e., create catalogs of entries that users could query, and generate automatic summaries of their knowledge bases (or selected parts of them) to give an end-user or expert system developer an overview of the knowledge they contain.

Automated Validation and Verification. As information is added to a knowledge base, a tool could verify that the new facts are not in conflict with others already in the knowledge base. If a fact does conflict, the tool should notify the developer, show which facts conflict, and allow fixing the discrepancy. If an expert system permits conflicting knowledge, the tool should still optionally notify the developer or keep a log of conflicts. A tool might also be able to check a knowledge base for semantic consistency (as defined by the user), for example, detecting rules that can never fire.

Retention of Test-Case Data. A tool should allow for storing test cases and their solutions as a system is being developed. As the knowledge base evolves, it can be tested automatically against the stored test cases to verify its correctness.

Change Logs. A tool should keep records of changes to the knowledge base, remembering who made the change, when it was made, and why. If discrepancies arise, developers can ascertain who made the changes and why and can reconstruct previous states of the knowledge base when necessary.

Software Engineering Aids

The participants expressed a number of desired extensions that can be thought of as software engineering aids:

- More explicit control over control
- Better documentation
- Debugging aids
- Quality assurance

More Explicit Control Over Control. Using some tools, the only way to affect the flow of control of an expert system is by reordering rules in the knowledge base. It is hard to express intentional orderings in this way, and adding a rule requires an understanding of the current ordering and the possible effects of changing it. It would be preferable in many cases to provide explicit control over this control flow.

Better Documentation. Tools need formal specifications, semantic descriptions, and good documentation. Often a tool user needs to enhance a tool, which is very difficult without complete documentation. To take full advantage of a tool, a user must have access to all the necessary information about that tool.

Debugging Aids. Debugging aids should be associated with each of the tool's features. For example, if the tool offers user-directed explanation, it should also offer a debugging aid for user-directed explanation. Since an expert system is a software product, all parts of it must be fully debugged for it to work properly and reliably.

Quality Assurance. Tool users need to know how rigorously a product has been tested before it is distributed. If a new version of a tool is released with possible bugs, this fact should be explained to users. Ideally, tools should go through test suites to insure that they have no glaring bugs. Quality assurance testing of this kind would be highly reassuring to users.

Tool vendors vary in the levels of testing they perform before releasing a new version of a tool. Users felt that a vendors' policies toward releasing software with bugs should be made explicit, so that users will know what to expect. Many users do not mind getting a new release that is not fully debugged if it has new features that they are eager to use, provided the vendor is responsive to bug reports and is willing to work with users to solve problems quickly.

Delivery Support

Workshop participants listed a number of desiderata related to tool support for delivering finished expert systems:

- Modifications for the delivery environment
- Assistance in generating efficient systems
- Real-time support and first-fit search
- Support for integration
- Support for human interaction
- Ability to selectively watch reasoning processes

Modifications for the Delivery Environment. Certain features of a tool that are useful during development may not be required in the delivered expert system. In fact, there may be features of a tool that should be kept hidden from end-users (e.g., debugging aids, explanations that use internal representations, or the ability to change the knowledge base). It should be possible to disable these tool features when fielding an expert system. If disabling certain features saves significant amounts of time or memory, it may also be useful to allow disabling them during development.

Assistance in Generating Efficient Systems. Performance is often a major issue in expert system design. It would be useful for a tool to provide performance analysis in the development environment to let the developer know whether it will perform satisfactorily. Such analysis would indicate the best areas for optimization in the fielded expert system. It might even be possible to compile into a faster language or to use more sophisticated optimization techniques prior to fielding.

Real-Time Support and First-Fit Search. Some applications require that an expert system perform in real-time. Since the necessary reasoning cannot always take place in the available time, this requires ways of constraining the inferencing mechanism for timeliness.

Similar mechanisms (i.e., constraining inferencing on the basis of resource consumption) would allow a form of "first-fit" search in which a system would reach an uncertain initial conclusion and would proceed on the basis of this conclusion. Subsequent reasoning could later replace this initial conclusion with one of greater certainty, in which case the system would interrupt itself and backtrack using the more certain conclusion.

Support for Integration. Integration was a major issue throughout the users' workshop. Participants noted that considerable effort was required to perform the integration necessary to implement their expert systems. They felt that tools should support integration with multiple knowledge sources, DBMSs, sensors, and effectors.

Support for Human Interaction. Most expert systems interact with human users. Support for user interface design is therefore a crucial requirement for a tool. For example, explanation should use the natural vocabulary and terminology of the expected users; this may require support for text or graphics, depending on the user community. The level of expertise of the expected users should also be considered: results must be presented in a way that will be readily apparent and easily interpreted. Tools should allow building interfaces that exploit a human's multi-processing capabilities, using a combination of text, graphics, and sound.

Furthermore, tools should support building interfaces that allow users to choose the level and form of output or explanation they receive. For example, different users may want different explanations that show the text of the rules that fired, or the first principles of those rules, or graphical representations of the tree of rules considered. Whereas an expert system developer may want to know how the system arrived at a conclusion, an end-user may want to know why the conclusion was true in a particular case. Tools should provide (or allow building) multiple models of users, covering a wide range of end-users and developers.

Ability to Selectively Watch Reasoning Processes. It is often useful for an expert system developer or end-user to observe some of the reasoning that leads an expert system to a particular conclusion. If the system is being used as an interactive aid, the user may want to see what leads it to a conclusion before acting on its advice.

Appendix A

LIST OF PARTICIPANTS

EXPERT SYSTEM TOOL DEVELOPERS' WORKSHOP

Richard Fikes
Intellicorp
1975 El Camino Real West
Mountain View, CA 94040-2216

Steve Hardy
Teknowledge
P.O. Box 10119
Palo Alto, CA 94303

Lowell Hawkinson
LISP Machines, Inc.
1000 Massachusetts Ave.
Cambridge, MA 02138

David Hornig
Carnegie Group
650 Commerce Court
Station Square
Pittsburgh, PA 15219

Anthony Magliero
Software Architecture
& Engineering
1600 Wilson Boulevard
Suite 500
Arlington, VA 22209

Steve Pardue
Radian Corporation
8501 Mo-Pac Blvd.
P.O. Box 9948
Austin, TX 78766

Charles Riese
Radian Corporation
8501 Mo-Pac Blvd.
P.O. Box 9948
Austin, TX 78766

Mark Wright
Inference Corporation
5300 West Century Blvd.
7th Floor
Los Angeles, CA 90045

RAND Corporation Participants

Iris Kameny

Jody Paul

Susan Pond

Jeff Rothenberg

Don Waterman

Dean Schlobohm (Consultant)
Greene, Radovsky, Maloney & Share
1 Market Plaza
Spear St., Suite 3200
San Francisco CA 94105

EXPERT SYSTEM TOOL USERS' WORKSHOP

Richard Adler
The MITRE Corporation
Burlington Road
Bedford MA 01730

Scott Austin
Northrop Corp., Anaheim
Electro Mechanical Division
500 East Orangethorpe Ave.
Anaheim CA 92801

Brian Baxter
United Fire & Casualty Life Ins.
118 Second Avenue, S.E.
P.O. Box 4909
Cedar Rapids IA 52407

Kirstie Bellman
The Aerospace Corporation
MS: M1/102
P.O. Box 92957
Los Angeles CA 90009-2957

Mark Bramlette
Lockheed California Company
Dept. 7253
Building 180, Plant B1
P.O. Box 551
Burbank CA 91520

David Buffo
Digital Equipment Corp.
Mail Stop: HL02-3/C10
77 Reed Road
Hudson MA 01749

John Davidson
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean VA 22102

Frank Edden
EATON Corporation
AIL Division
Commack Road
Deer Park NY 11729

Richard Feifer
UCLA Center for the Study of Evaluation
145 Moore Hall
Los Angeles CA 90024

Ken Gilbert
Digital Equipment Corp.
Mail Stop: HL02-3/C10
77 Reed Road
Hudson MA 01749

Brent Hadley
Boeing Military Aircraft Co.
17501 Southcenter Parkway
Tukwila WA 98188

Ted Kitzmiller
Boeing Computer Services
P.O. Box 24346
MS7L-64
Seattle WA 98124

Steven C. Laufman
Battelle Northwest
P.O. Box 999
MS: Math Bldg.
Richland WA 99352

Mike Liebhaber
University of Kansas
1043 Indiana
Lawrence KS 66044

Miguel Marin
Institut de Recherche d' Hydro-Quebec
IRAQ, VARENNES
Quebec JOL 2PO
CANADA

John Marinuzzi
Los Alamos National Laboratory
Knowledge Systems Laboratory (Rm. 17)
1900 Diamond Drive
Los Alamos NM 87544

Ted Markowitz
American Express Corp.
Systems & Technology
American Express Tower, WFC
New York City NY 10285-4545

John Mitchiner
Sandia National Laboratory
Computer Sciences Department
P.O. Box 5800
Albuquerque NM 87185

Ken Modesitt
California State University
Computer Science Department
18111 Nordhoff St.
Northridge CA 91330

Michael Morrison
Texas Instruments, Inc.
Manager, Knowledge Engineering
Data Systems Group
P.O. Box 2909
MS 2195
Austin TX 78769

Laurie O'Connor
Hughes Aircraft Co.
Member of Technical Staff
Autonomous Systems Section
Artificial Intelligence Center
Research Laboratories
Bldg. 150, M/S A600
23901 Calabasas Road
Calabasas CA 91302

Brad Pollock
  Cedars-Sinai Medical Center
  Department of Cardiology
  8700 Beverly Boulevard
  Los Angeles CA 90048

Lt. David Rosenberg
  Airforce Space Division/CFPF
  P.O. Box 92960
  Los Angeles AFS
  Los Angeles CA 90009-2960

Steve Rosenberg
  Hewlett Packard, Palo Alto
  1501 Page Mill Rd.
  Palo Alto CA 94304

Ron Siemens
  Ford Aerospace
  1260 Crossman
  MS: S37
  Sunnyvale CA 94086

Ken Strzepek
  University of Colorado
  Civil Engineering Department
  Campus Box 428
  Boulder CO 80309-0428

George Tetterton
  General Dynamics
  Data Systems Division
  P.O. Box 748
  Mail Zone 5305
  Fort Worth TX 76101

Larry Tieman
  American Airlines
  P.O. Box 582809, MD 353
  4000 North Mingo Road
  Tulsa OK 74158-4560

Hao N. Vu
  First Interstate Services
  Systems Consultant
  First Interstate Services Corp.
  P.O. Box 935
  El Segundo CA 90245

Robin Webster
Rockwell
% Anne Stanley
1524 Welldow Lane
Fullerton CA 92631

Howard Weiser
Arthur Andersen & Co.
33 West Monroe Street
Chicago IL 60603

Mark Young
Hughes Aircraft Co.
Radar Systems Group
MS: RE/R11/8003
P.O. Box 92426
Los Angeles CA 90009

RAND Corporation Participants

Iris Kameny

Jim Kipps

Jody Paul

Susan Pond

Jeff Rothenberg

Marcy Swenson

Dean Schlobohm (Consultant)
Greene, Radovsky, Maloney & Share
1 Market Plaza
Spear St., Suite 3200
San Francisco CA 94105

Appendix B

QUESTIONNAIRES

EXPERT SYSTEM TOOL USER QUESTIONNAIRE I

The following questions are intended to characterize your background and experience using expert system (ES) tools. Please attach your answers and return this questionnaire before October 1. Your answers can be informal, but try to make them complete enough to include relevant context. If you are involved in multiple projects, please provide a separate set of answers for each. If you prefer you can respond by electronic mail to jeff@rand-unix.

YOURSELF

1. Please include your name, affiliation and address, phone number and extension, and electronic mail address (if any).

TARGET TASK, DOMAIN & USERS

2. Within the limits imposed by security or proprietary constraints, describe your ES task and its intended domain of application.

3. What are the major issues and concerns involved in developing this application?

4. What was the motivation/justification for applying AI/ES technology to this task?

5. Is your ES being developed

   o in the line of research?
   o as a prototype?
   o as an in-house tool?
   o for developing custom products?
   o as a commercial product?

6. What is the scale of your problem? (If possible, give some approximate idea of the size of your expected effort, e.g., anticipated person years, number of rules, size of database, etc.)

7. Characterize your expected end users (e.g., domain experts? programmers? computer-naive professionals? non-technical laymen?).

8. How would you characterize your ES task in terms of "standard" AI tasks such as diagnosis, planning, etc.?

9. Characterize the hardware/software of your development environment and, if different, your target delivery environment.

DEVELOPMENT TEAM

10. Describe the members of your development team in terms of their background and ability (i.e., characterize how much experience your group has had with ES development, ES tools, AI, software engineering, user-interface design, applications programming, etc.).

11. What is your source of domain expertise?

12. To what extent do you have end users involved in the development effort?

EXPERT SYSTEM TOOLS

13. What ES tool(s) are you using? (If an in-house tool, please describe; if commercial, name product and vendor.)

14. What other ES tool(s) did you consider?

15. What evaluation criteria were used to select the ES tool(s)?

16. How long did it take to learn to use the ES tool(s) and what difficulties were encountered along the way?

17. How would you characterize the strengths and weaknesses of the ES tool(s) for your task?

18. Describe your overall reaction to using ES tool(s) based on your experience.

EXPERT SYSTEM TOOL USER QUESTIONNAIRE II

This questionnaire should be read and answered as completely as possible before the start of the workshop November 3. The questions given here will form the basis for organizing the first day's sessions. We will begin the workshop with a presentation and discussion in which we will resolve any ambiguities that arise as you try to answer these questions. We will then allow time to revise and refine your answers before collecting them, copying them and distributing them to the other attendees for use in working groups. If some of your answers are proprietary to the extent that they should not be distributed to other group members, you may want to generate two versions of the questionnaire: a complete version for use only by RAND project members and a "sanitized" version for use by other attendees.

{name}

The following questions are intended to give us an in-depth look at your background, to characterize your views about evaluating expert systems, and to familiarize you with some terminology which will form a vocabulary for our discussions. We held a workshop June 26-27, 1986, for commercial expert system (ES) tool developers, and we present here some of the ideas for ES tool evaluation that arose from that workshop. Our intent is to share the results of that workshop, and to validate, refine and extend the ideas that originated there.

The first workshop identified five areas for ES tool evaluation:

1) CONTEXTS (e.g., prototyping an ES, using an ES)
2) PROBLEM CHARACTERIZATION (e.g., domain (medical), type (planning), complexity)
3) CAPABILITIES (e.g., inference, explanation, knowledge acquisition)
4) METRICS (e.g., cost, flexibility, portability)
5) EVALUATION METHODS (e.g., questionnaires, comparisons, interviews)

These form 5 dimensions for evaluating an ES tool. Consider the 5-dimensional space in which each point represents the cross product: a context X a problem X a capability X a metric X a method. For example, using an expert system X thyroid disorder diagnosis X explanation X flexibility X comparisons.

The resulting space also has meaningful subspaces, such as context X metric (e.g., prototyping environment X portability), and capability X metric (e.g., knowledge acquisition X flexibility).

In this questionnaire, we will ask about your particular ES development project, discuss each of the 5 dimensions above, and ask some questions about each one.

YOUR EXPERIENCE WITH EXPERT SYSTEMS TOOLS

The following questions concern the ES you are currently developing, rather than the tool you are using to develop it. In order to relate the size of your problem to the size of your effort, we would like to know how long the effort has been going on, who has been working on it, and how large the system is.

1. If your answer would differ from that submitted in the first questionnaire, then please describe your ES task and its intended domain of application (within the limits imposed by security or proprietary constraints).

2. Are you building:

   - an expert system application?
   - an enhanced environment for building expert systems?

3. Is the objective of your ES task to:

   - do research?
   - develop a prototype?
   - develop an in-house product?
   - develop a one-off product for a customer?
   - develop a commercial product?

4. Which stage of development best describes your ES task at this time?

   - experimenting/looking at tools
   - prototyping/demonstrating feasibility
   - developing
   - fielding
   - delivering/maintaining

5. If your ES task is to build an ES application, then please fill in the following table that shows development stages vs. types of people involved. In the top row, show what percentage of each stage is complete, and in the remaining rows, fill in the number of full-time equivalent people that participated in that particular stage of development. For instance, if one systems analyst worked full-time, and two systems analysts worked 1/3-time on the development stage, then put 1.6 in the corresponding box. Add rows as necessary.

TABLE 1	Initial Planning/ Problem Scoping	Knowledge Acquisition	System Building	Debugging	Fielding	Maintenance/ Enhancement
Percent Complete						
Knowledge Engineers						
Systems Analysts						
Project Managers						
Domain Experts						
Programmers						
AI Programmers						
others (e.g. Graphics Programmers) Please identify:						

6. To characterize the size, complexity, and coverage of your ES task, please fill in the following table, picking the most appropriate unit(s) of measure for your system. For example, if your ES task has approximately fifty rules concerning control information, enter 50 in the upper left box.

TABLE 2

How much of the system is:	rules	frames	objects	behaviors	Klips	lines of code
control information						
user interface						
domain information						
fielding environment						
explanation						

THE FIVE DIMENSIONS FOR ES TOOL EVALUATION

The five dimensions for tool evaluation are: contexts for evaluation, problem characterization, tool capabilities, metrics, and evaluation methods.

The questions in this section are about evaluating ES tools in general, and not the specific ES tool you have used. We are looking for evaluation criteria that will be universally applicable.

A. CONTEXTS FOR EVALUATION

There are several different ways an ES tool is used. It can be used for rapid prototyping, for development or for fielding mature systems.

These are some of the relevant contexts for evaluation:

Abstract

the tool as a conceptual framework and a piece of software implementing that framework, in the absence of any more specific context. This emphasizes the quality and usability of the tool and its interface.

Prototyping environment

the tool as used in prototyping an ES. This emphasizes the tool's facilities for quickly trying alternative approaches and representations.

Development environment

the tool as used in developing a mature ES. This emphasizes the tool's suitability for development of software, including its debugging facilities, configuration control, etc.

Execution/fielding environment

the tool as it affects the delivery of a finished ES and the performance and interface of that ES. This emphasizes the tool's facilities for porting from the development environment to the delivery environment and for maintenance, support and performance in the delivery environment.

Life cycle of the tool

the tool's usability over an extended period of time. This emphasizes the ability of the tool to evolve over time and the support provided by its vendor.

1. Please add contexts that you feel are missing.

2. Do you feel any of the contexts are more or less important than any others, if so then why?

B. PROBLEM CHARACTERIZATION

Just as one would not evaluate a car without considering where it will be driven (e.g., in a traffic-ridden city, on sand dunes, or in snow) it makes no sense to evaluate an ES tool without considering what it will be used for. We would like to find not only which ES tools are generally useful, but which types are useful for certain applications, e.g., what qualities would one look for in an ES tool to use for geological simulation?

Below are four ways of categorizing problems, and for each categorization, some problem types.

Are there other useful ways of categorizing problems?

Would you add any types for any of the given categories?

1. *Problem domain examples:*

chemistry	mathematics
electronics	medical
geological	military science
information management	physics
legal	space technology
manufacturing	

2. *Problem types:*

classification	monitoring
control	planning
debugging	prediction
design	repair
diagnosis	simulation
instruction	

3. *Complexity*
    - Scope
    - Size

4. *Development team characteristics*
    - AI experience
        - Knowledge engineers
        - AI programmers
    - Computer science background
    - Programming experience
    - Domain experts
    - Number of people

C. CAPABILITIES OF A TOOL

Capability refers to a tool's ability to readily support certain aspects of its applications. A capability is supported through features. A tool must have at least one feature for each capability it realizes and a given feature may support more than one capability.

For example, the capability of shifting gears allows an automobile to change speed ranges. Two different features that support shifting gears are manual transmission and automatic transmission. As another example, the capability of grasping an object allows one to reposition the object. Two different features that support grasping objects are hands with opposable thumbs, and magnets.

The following list of capabilities and features was developed in the first workshop, and we would like your validation and input. Please edit and comment on the list.

1. *Arithmetic processing*

   Does a tool provide a full range of arithmetic operators and numbers (e.g., long integers, floating point)?

2. *Certainty handling*

   Does a tool have built-in methods for representing and propagating certainty?

   Can these mechanisms be augmented, modified, or defeated?

   Supporting features:
    - Built-in representation methods like Bayesian, likelihood ratios
    - Built-in propagation methods
    - Fuzzy logic

3. *Concurrency*
Supporting features:
- Distributed processing
- Parallel processing

4. *Consistency checking*
Does a tool check for syntactically correct but internally inconsistent or implausible knowledge?

Is consistency enforced by a conceptual formalism that doesn't admit inconsistent knowledge?

Is consistency enforced by built-in mechanisms and checks?

Is consistency enforced by requiring the user to follow certain conventions?

5. *Development documentation*
Does a tool allow recording assumptions and rationale about the knowledge during the development process (e.g., annotations on code and data structures)?

Can this knowledge be interpreted by the system?

6. *Explanation*
Does a tool have explanation?

Can the explanation mechanism be augmented, modified, defeated?

7. *External access*
Can a tool directly access external programs, operating systems and databases during execution?

Can a tool access special hardware (e.g., sensors, effectors)?

8. *Inference and control techniques*
Supporting features:
- Iteration
- Recursion
- Forward chaining
- Backward chaining
- Inheritance; multiple hierarchies or lattices
- Other relations (such as part/whole, nearby)
- Dependency structures and dependency-directed backtracking
- Multiple worlds or viewpoints
- Demons or triggers

9. *Internal access*
Can a tool access the underlying language in which it was programmed (e.g. Lisp)?

Does this access work consistently in the development and delivery environments?

10. *Knowledge acquisition*
Does a tool provide sophisticated run-time acquisition?

Does a tool provide sophisticated acquisition for development? For both declarative and procedural knowledge?

Does a tool support the acquisition of domain specific reasoning or control methods?

Does a tool support the acquisition of general reasoning or control methods, e.g., planning techniques, common sense reasoning?

Does a tool support the acquisition of knowledge needed for explanation?

Does the system have special methods for initializing the knowledge base?

Does a tool derive its own rules directly from the data?

11. *Knowledge-base editing*
Does a tool have good text editing and formatting?

Does a tool have good structure editing and formatting?

12. *Meta-knowledge*
Does a tool have meta-knowledge about control?

Does a tool know the limits of its capabilities?

13. *Optimization*
Supporting features:
  Intelligent lookahead
  Result caching
  Rule compilation

14. *Presentation I/O*
How simple, visually effective and efficient is the interface to both the system builder and the end-user?

How easily can the interface mechanisms be augmented, modified, or defeated?

Can multiple, overlapping windows be created using the knowledge engineering language?

Supporting features:
- Textual
- Graphical
- Windowing
- Spreadsheet
- Forms
- Mouse
- Light pen
- Touch-sensitive screen

15. *Representation techniques*
Supporting features:
- Rules
- Frames
- Procedures
- Logics
- Objects
- Triggers, demons
- Blackboard
- Temporal representation
- Spatial representation
- Simulation
- Planning

D. METRICS

Given then capabilities and features we would like to evaluate, and the contexts to evaluate them in, it would be helpful to have some metrics with which to measure them. We could just use "great," "OK," and "bad," and say an ES tool rates "OK" in inference techniques in the delivery environment; but an "OK" doesn't really give us a very accurate measure. We would like to be more specific, and say that the inference techniques in the delivery environment are very flexible, not very powerful, but very efficient. Different metrics measure different qualities of a tool, expressed in appropriate units.

The following is a list of metrics that seem applicable to evaluating ES tools. Please add to the list any other metrics that you think are appropriate for ES tool measurement, and cross out any that you think are inappropriate.

METRIC	EXAMPLE
Adequacy	of a tool or its feature to a given task or range of tasks.
Availability	in the face of downtime or maintenance requirements.
Breadth	applicability to a wide range of tasks, domains, etc.
Clarity	of concepts, mechanisms, interface, etc.
Cognitive efficiency	number and complexity of concepts, techniques, special cases, etc. that a user must keep in mind to use the system effectively, relative to the power they provide.
Coherence	simplicity and generality of tool features and mechanisms.
Completeness	with respect to a given task or range of tasks.
Congruence	Concepts that are OR SHOULD BE similar (different) to the intended user should map to similar (different) things within a tool.
Consistency	of features or mechanisms within a tool.
Controllability	of a tool or its features by its intended users.
Cost	direct costs (hardware & software) and indirect costs (support, maintenance, etc.)
Defeatability	the ability to override features or functions of a tool.
Ease of Use	a subjective measure, but given in absolute terms (as opposed to cognitive efficiency which is relative to power).

Efficiency	measured usage of time and memory.
Extensibility	ability to enhance existing capabilities or features or add new ones.
Flexibility	ease of extensibility, provision of alternative ways of representing or solving problems, etc.
Integration	of features, representations, sub-systems etc., both within a tool and with external languages, existing software, databases, hardware, etc.
Maintainability	ease of access to underlying mechanisms and support for fixing bugs, changing limits, modifying built-in behavior.
Modularity	of representation and control
Philosophy	attitudes, preferences, tradeoffs etc. embodied in a tool.
Portability	of both development and delivery systems; compatibility of interface, limitations, etc. across machines.
Power	Overall leverage compared to an underlying language/system, within the scope of applicability of a tool. Power is used here to imply depth rather than breadth.
Reliability	is a measure of the success with which a tool conforms to its specifications, i.e., does it work right all the time?
Responsiveness	the speed and appropriateness of a tool's reaction to the user.
Robustness	the ability of a tool to continue working in the face of unexpected inputs, hardware or software failures, malevolent attacks, etc.
Scalability	the ability to handle both large and small problems, and especially to allow starting small and growing without having to redesign.
Sophistication	relative to an underlying language/system or the state of the art.
Subsetability/Separability	Can some parts of the system be used independently of others? Is it necessary to learn and "pay for" all of a system's complexity even to do fairly simple things?

Usability	Any given feature or aspect of a tool can be evaluated with respect to how usable it is in practice by real users.
Learnability	Any given feature or aspect of a tool can be evaluated with respect to how easy it is for real users to learn.

E. EVALUATION METHODS[1]

There are many ways of evaluating something; hotels are rated with stars, Consumer Reports rates things with big grids of data and movies are rated by critics, who often just talk subjectively about the films. A combination of several methods would provide more information, but there is a threshold after which more information is confusing. We need to choose the methods which would be most helpful in evaluating ES tools.

Below are some methods of evaluation. For each method, please assign it a usefulness rating from 1 to 5, 1 being "practically worthless," and 5 being "highly useful."

1. *Comparisons*

A. Compare each tool with some sort of "baseline" tool. This would necessitate choice of a baseline tool.
Rating:

B. Compare each tool with some sort of ideal tool. This would necessitate a definition of the ideal tool.
Rating:

C. Compare tools with one another (similar to Consumer Reports, but more in-depth).
Rating:

[1]Note that the term "Evaluation Methods" was subsequently changed to "Assessment Techniques."

2. *Benchmarks*

By "benchmarks" we do NOT mean timing tests of standard algorithms, but rather special problems developed to test the capabilities of an ES tool. Certain problems would be offered in non-implementation-specific terms, and would be solved by vendors offering ES tools or by prospective users. Solutions for each tool would be published, along with the time required to implement each solution, and the resulting system size, etc. Solutions could also be evaluated on the basis of style, clarity and conceptual cleanliness.

D. Small benchmark problems can be solved in hours or days, they consist of an informal statement of the desired capability, a specific problem fragment for testing the capability, and the role this capability plays in solving the problem. (For an example of a small benchmark problem, see attachment A.)
Rating:

E. Large benchmark problems can be solved in weeks or months, they consist of a detailed description of the problem being addressed, a checklist of things to which the implementor must respond during the development process (like how hard was "x" to implement, or how long did "y" take you), and follow-up interviews to obtain subjective evaluations. An example of a large benchmark problem is that of locating and diagnosing a spill in a chemical plant. It is presented in detail in "Building Expert Systems," (Hayes-Roth, Waterman, and Lenat (eds), Addison-Wesley, 1983).

F. If you can suggest any applicable small or large benchmarks, please describe them below.

3. *Case Studies*

G. Controlled recording of an ES development effort, in an attempt to capture the relevant aspects of the tool being used.
Rating:

4. *Library of Expert System Efforts*

H. Information about ES development efforts, organized in a database that can be searched to find projects similar to a proposed project.
Rating:

5. *Interviews*

I. Interview people who have completed long-term development efforts.
Rating:

6. *Questionnaires*

J. Collect information from various users of a tool, keep on file in an accessible location.
Rating:

7. *An Expert System for Tool Evaluation*

K. Develop an ES that knows about previous efforts and what qualities were useful for a particular type of application, and can perform consultation and diagnosis for a potential user about what type of ES tool would be most useful.
Rating:

ATTACHMENT A
(of Questionnaire II)

An Example of a Small Benchmark Problem

Experts may organize their domain knowledge around taxonomies, hierarchical structures in which properties of an entire class can be stated just once and then "inherited" by members of that class or any subclass.

For example, all squares are rectangles and all rectangles are geometric figures. The area of any rectangle can be computed by multiplying its height by its width. The width of a square equals its height.

Represent this knowledge and use it to compute the area of a square called "object-22", with a height of five units. Rectangle height must be a number. Extend the representation so the system will object to non-numeric heights for rectangles.

Appendix C

BENCHMARKS

SMALL BENCHMARK PROBLEMS

Contributed by Dean Schlobohm

This memo describes what is meant by a "small benchmark" for use in evaluating expert system tools ("tools"). It also describes potential capabilities which may be able to be evaluated using small benchmarks. Finally, it gives two examples of portions of a small benchmark.

I. Description of a Small Benchmark

Each small benchmark would assist in evaluating one or more functional capabilities of a given a tool. A small benchmark should be able to be implemented in a given tool within a relatively short period of time, ranging from a few person-hours to several person-days, not including the time required to fill out the evaluation questionnaire. Each benchmark should contain the following:

1. A description of the capability or feature being tested by the benchmark
2. A description of the type of problems for which the capability or feature being tested may be important and why
3. A description of a specific small problem for testing the capability or feature
4. A checklist and/or a set of questions which the person implementing the benchmark in a given tool (the "tester") could use in evaluating the tool with respect to the benchmark

It is assumed that most of these small benchmarks will test just one capability or feature. If they test more than one, please respond to (1) and (2) above for each capability or feature being tested.

Each of these will now be discussed in more detail.

A. Capability or feature being tested

This should be a short description of the capability or feature of a tool which the small benchmark will assist in evaluating. Since it is expected that some of the benchmarks will be implemented by a "novice" tester (i.e., a domain expert with little computer knowledge or a programmer with little AI knowledge), the description should be clear and understandable to such persons.

The description might contain examples of the feature taken from existing expert systems. These examples should not contain the actual code since the code could influence how the tester attempts to implement the benchmark in a given tool.

B. The types of problems for which the capability or feature being tested may be important and why.

One of the major issues in evaluating tools is the determination of what capabilities are necessary (important, desireable) in order to create a commercial expert system of a given type in a given domain. A "feature list" will be of little use unless a person has a method for determining which tool features and capabilities are needed to solve his or her problem.

The types of problems could be: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, tutorial and control systems. Other suggested types would be appreciated.

We realize that the above types of problems may be of little help in matching tools to tasks since most actual problems problems will contain aspects of several types. We would appreciate your ideas on whether there is a classification of the types of problems which can be matched against tool capabilities or features to assist in evaluating whether a given tool may be effectively used to solve a given problem.

C. A specific small problem for testing the capability or feature.

The benchmark should be a description of some small problem, the solution of which could require the use of the capability or feature being benchmarked. In order not to influence the tester, the problem must be described in general terms and not in some implementation.

The tester should be required to first attempt to solve the problem by using an implementation which tests the capability or feature being benchmarked. However, the tester should be encouraged to also solve the problem using the method "best" supported by the tool. For example, although the problem might be designed to test the use of backtracking in a particular tool it may be more natural (efficient, etc.) to solve it using some other technique. The tester should be encouraged to evaluate the tool with respect to both methods.

D. A "checklist" and/or a set of questions which the tester can use in evaluating the tool with respect to the benchmark.

One of the reasons for using small benchmarks is that the tester will not have a lot of time to spend implementing them. Thus, to the extent possible, a checklist should be prepared to assist the tester in evaluating a given tool with respect to the benchmark. For example, the checklist could be of the form:

How would you rate the tool with regard to the following issues?

	Inferior	Poor	Fair	Good	Excellent
1. The ease with which the benchmark could be implemented in the tool					
2. The naturalness of the representation the tool provided for the problem					
. . .					

There should also be a set of questions asking the tester for his or her overall subjective evaluation of how effective the tool solved the benchmark problem. Finally, there should be questions relating to memory, speed, etc. which require objective answers.

Each of these questionnaires should be designed for a specific benchmark. For example, the questions for evaluating backward chaining may be different from those for evaluating windowing capabilities.

It may be appropriate to have a series of questionnaires to be completed at different times during the implementation of the benchmark. Alternatively, the tester may be presented with questions which require them to provide evaluations over time (i.e., keep a diary of the implementation process).

One of the initial purposes of creating and implementing small benchmarks is to obtain more knowledge about how to evaluate knowledge engineering languages. As a result, we expect that some of the benchmark methods may not be as effective as others at testing the target capability or feature. Thus, the testers should be presented with questions which permit them to evaluate whether the benchmark problem adequately assists in evaluating the capability or feature.

The tester should also be given the opportunity to suggest other methods and/or problems which can be used to evaluate a given capability or feature. This will allow us to create a larger library of small benchmarks and other methods to evaluate tools.

II. Potential Capabilities To Be Evaluated

We believe that the following features and capabilities constitute a partial list of those which can be evaluated (to some degree) by using small benchmarks:

1. Rules
    - Forward chaining
    - Backward chaining
    - Data driven reasoning
    - Blackboard reasoning
    - Representation of meta knowledge
    - Other inference and control techniques

2. Frames
    - Inheritance
    - Multiple inheritance

3. Object-oriented programming
    - Message passing

4. Certainty handling
    - Built-in features
    - User creation of alternative methods

5. Explanation capabilities
    - Built-in features
    - User creation of alternative methods

6. End-user interfaces
    - Menus
    - Windows
    - Data type checking
    - Graphical facilities
    - English understanding and generation

7. Arithmetic capabilities
    - Floating point arithmetic
    - Built-in functions for financial calculations

8. Automatic Knowledge Acquisition
    - Induction

9. Debugging facilities
    - Tracing
    - Graphical representations
    - Consistency checking of the knowledge base

III. One Example of a Small Benchmark

The following is a portion of a small benchmark. It should be noted that much of the benchmark still needs to be completed.

A. Features Being Tested

This benchmark will help evaluate the tool with respect to the following features: frame mechanisms and inheritance, windows, menus, English generation, and the creation of explanations.

B. Type of Problems For Which the Features May Be Important

The features evaluated in this benchmark are important in solving problems which require "user friendly" interfaces. Also, problems which are hierarchical in nature can usually be more easily represented in frame systems.

C. The Problem

Many expert systems require the capability of providing explanation to end-users. Furthermore, knowledge in many domains is hierarchical in nature. In order to test these features (among others), please solve the following problem in your tool:

You are to create a program which will be able to explain various predetermined terms and concepts in text that is presented to the end-user. The knowledge should be stored in the following hierarchy:

```
                        TRUST
                          |
                          |
                  ------------------
                  |                |
              REVOCABLE       IRREVOCABLE
                TRUST            TRUST
                                   |
                                   |
                      ---------------------------
                      |                         |
                    BYPASS                   MARITAL
                    TRUST                   DEDUCTION
                      |                       TRUST
                      |                         |
              ----------------                  -
              |              |                  |
         ALL INCOME     SPRINKLING             QTIP
           TRUST          TRUST               TRUST
```

The following information should be stored with each of the elements in the hierarchy.

BYPASS TRUST

1. Explanation -- "A bypass trust is established for NAME-OF-SPOUSE so that the property in the trust will not be taxed in NAME-OF-SPOUSE's estate."

2. Explainable terms -- PROPERTY, ESTATE

IRREVOCABLE TRUST

1. Explanation -- "The trustor retains no power to amend or revoke the trust."

2. Explainable terms -- TRUSTOR

In the above, NAME-OF-SPOUSE is a variable which should be filled in with the actual spouse's name at run time.

The explanations should be of the form:

```
                        -----------------------------------------------------
----------------------- | EXPLANATION OF BYPASS TRUST                        |
| EXPLANATION MENU    | |                                                   |
----------------------- |    A bypass trust is an irrevocable trust.  A     |
|                     | | bypass trust is established for Mary Jones so     |
|                     | | that the property in the trust will not be taxed  |
|SPRINKLING TRUST     | | in Mary Jones' estate.                            |
|ALL INCOME TRUST     | |                                                   |
|IRREVOCABLE TRUST <==| | Examples of a bypass trust are:                   |
|PROPERTY             | |                                                   |
|ESTATE               | |    all income trust                               |
|                     | |    sprinkling trust                               |
|RETURN               | |                                                   |
|CHANGE LEVEL         | |                                                   |
|CONTINUE             | |                                                   |
----------------------- -----------------------------------------------------
```

```
                    ----------------------------------------------------
                    | EXPLANATION OF BYPASS TRUST                      |
                    |                                                  |
                    |   ----------------------------------------------------
---------------------   | EXPLANATION OF IRREVOCABLE TRUST                 |
| EXPLANATION MENU  |   |                                                  |
---------------------   |    An irrevocable trust is a trust. The trustor  |
|MARTIAL DEDUCTION  |   | retains no power to amend or revoke the trust.   |
|  TRUST            |   |                                                  |
|TRUST <==          |   | Examples of an irrevocable trust are:            |
|TRUSTOR            |   |                                                  |
|                   |   |    bypass trust                                  |
|RETURN             |   |    marital deduction trust                       |
|CHANGE LEVEL       |   |                                                  |
|CONTINUE           |   |                                                  |
|                   |--|                                                  |
|                   |   |                                                  |
|                   |   |                                                  |
---------------------   ----------------------------------------------------
```

In the above, the examples and the sentences, "A bypass trust is an irrevocable trust." and "An irrevocable trust is a trust." should be generated from the frame hierarchy. Only explanations unique to a given concept should be stored with that concept. Also, the name Mary Jones should be retrieved from a representation of the Client's spouse and inserted in the text just before presenting the explanation.

Selections from the Explanation Menu should be able to be made either by using a mouse or some keyboard input.

While the above problem suggests the use of a frame representation, you should also solve the problem in the manner best suited to the tool you are testing. You must not, however, use only "canned" text.

D. The Questionnaire

How would you rate the tool with regard to the following issues?

	Inferior	Poor	Fair	Good	Excellent
1. The ease with which the knowledge hierarchy was implemented in the tool					
2. The naturalness of the representation the tool provided for the probiem					
3. The speak/efficiency of the end-user interface					

Please elaborate:

	Inferior	Poor	Fair	Good	Excellent
4. The effectiveness of the benchmark problem as set forth above in testing the tool's ability to create user friendly interfaces					

Please elaborate:

5. How long did it take you to represent the trust hierarchy? (in person-hours of effort) ______________________

6. How long did it take you to create the overlapping windows? (in person-hours of effort) ______________________

7. Do you have a problem which could be used to evaluate a tool's ability to represent hierarchical knowledge? Is so, please describe.

IV. A Second Example of a Small Benchmark

A. Features Being Tested

This benchmark will help evaluate the tool with respect to the following features: frame mechanisms and inheritance, use of default reasoning, and the ability to deal with inconsistent data.

B. Type of Problems For Which the Features May Be Important

The features evaluated in this benchmark may be important in problems which are hierarchical in nature and may contain inconsistent data.

C. The Problem

In this benchmark, you are to add certain facts to a knowledge base and then ask questions about the knowledge. Please use the representation most suitable for the tool you are testing.

Add: Trucks are vehicles.
Add: Big red trucks are trucks.
Add: T1 is a big red truck.

Ask: Is T1 a vehicle?

Add: All big red trucks have color red.

Ask: What is the color of T1?

Add: All trucks have at least 4 wheels.

Ask: Does T1 have wheels?

Add: All vehicles have exactly one color.

Ask: Does T1 have color black?

Add: T1 has color green. (Inconsistent)
(Note: This should be disallowed.)

Retract: All vehicles have exactly one color.

Ask: What is the color of T1?

Add: The weight of big red trucks is typically 5000 pounds.

Ask: What is the weight of T1?

Add: The weight of T1 is 7000 pounds.

Ask: What is the weight of T1?

D. The Questionnaire

How would you rate the tool with regard to the following issues?

	Inferior	Poor	Fair	Good	Excellent
1. The naturalness of the representation the tool provided for the problem					
Please elaborate:					
2. The adequacy of the tool in warning you of the addition of inconsistent knowledge					
Please elaborate:					
3. The capability provided by the tool for browsing through the knowledge base					
Please elaborate:					
4. The effectiveness of the benchmark problem as set forth above in testing the tool's ability to represent hierarchical knowledge					
Please elaborate:					

5. Do you have a problem which could be used to evaluate a tool's ability to represent knowledge? If so, please describe.

SMALL BENCHMARK ON AIDS TO KNOWLEDGE ACQUISITION

Contributed by Radian Corporation

A. Description of the Feature

This benchmark can be used to evaluate expert system tools with respect to the aids they provide for knowledge acquisition.

B. Types of Problems

Aids to knowledge acquisition are important for any problem in which there is a large amount of rule-based knowledge.

C. The Problem

In this exercise you are to build up a knowledge base by adding pieces of information consecutively. The ultimate goal is to produce a system which can classify any closed four-sided figure (with no sides intersecting) correctly and specifically.

The information is presented in two forms. Declarative statements are embedded in bold in the instructions. Examples of four-sided figures are presented in picture form.

BEGINNING ASSUMPTIONS

Closed 4-sided figures with no sides intersecting
Definitions

parallelogram	- four sides with opposite sides parallel
right	- having at least one right angle
quadrilateral	- four sides

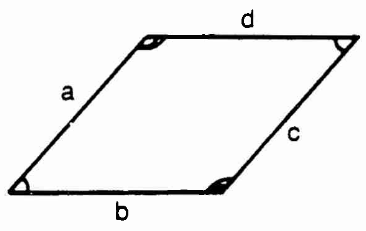
1. GETTING STARTED

Here is a list of attributes or properties considered relevant to the classification of four-sided figures.

a. whether all four sides are equal in length (equilateral)
b. the number of opposite sides which are parallel
c. the presence of a right angle
d. the presence of an interior angle of > 180 degrees
e. whether all interior angles are equal

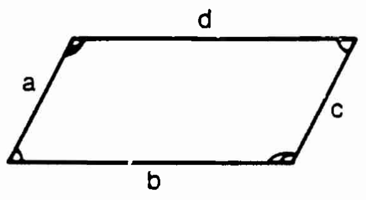
Examples of Four-Sided Figures

a=b=c=d (All sides equal)
a∥b, b∥d (4 sides parallel)



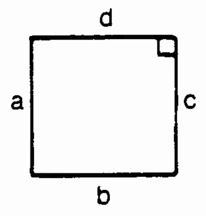
Rhombus

a=c, b=d, a≠b
a∥c, b∥d



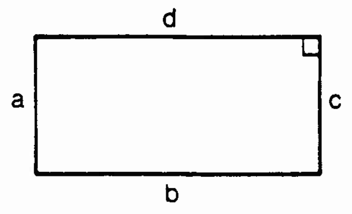
Rhomboid

a=b=c=d
a∥c, b∥d
90° angle



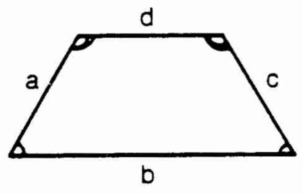
Square

a=c, b=d, a≠b
a∥c, b∥d
90° angle



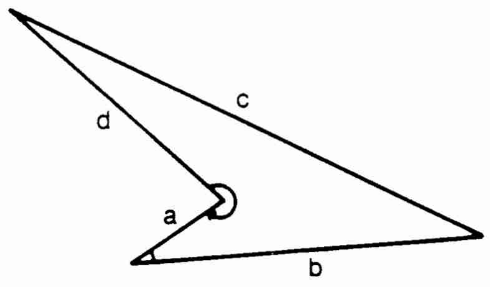
Rectangle

b∥d



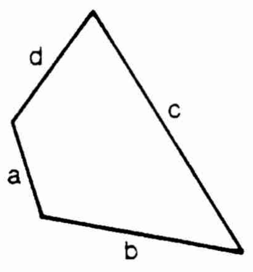
Trapezoid

a≠b≠c≠d



Concave Trapezium

a≠b≠c≠d



Convex Trapezium

10.86 25936

Enter:

> **An equilateral parallelogram is a rhombus A non-equilateral parallelogram is a rhomboid**

Can you create an executable system? If so, does the system ask about both properties (a and b) before naming the figure? Given the limited world so far, does it name it correctly?

2. CONTRADICTORY OR UNSPECIFIC INFORMATION

Enter:

> **A square is an equilateral paralellogram**

While the sentence is true, it is also true that a further property distinguishes a square from a rhombus, namely the presence of a right angle. The same attribute values, equlilateral and all sides parallel, now lead to different clas sifications, a rhombus and a square.

A tool can handle this in one of several ways. It may make you clarify the situation before letting you create an executable, it may wait until run-time to inform you, or it may never inform you. If you can run the system with just these rules, try to classify a rhombus and then a square. Does it always tell you the figure is a rhombus? a square? both? either? Is the classification wrong, right, or lacking in specificity?

Amend the information by adding the attribute of "rightness" (attribute c). Enter:

> **An equilateral right parallelogram is a square**
> **A non-equilateral right parallelogram is a rectangle**

Once you have added this information, test the system's behavior for all the figures defined so far. How does it perform? Was it easy to add the attribute?

3. SUBSETS AND SUPERSETS

Change the information on rhomboids:

> **A rhomboid may or may not have a right angle**

Again, while the sentence is true, it is the property of "rightness" which distinguishes rectangles from rhomboids. Different values of an attribute, the presence and absence of a right angle, now lead to the same classification, a rhomboid.

Again, a tool can tell you that the rectangle is a subset of the rhomboid (or the rhomboid is a superset of the rectangle) before run-time, during run-time, or never at all. If you can run the system, does it correctly and specifically classify a rectangle? a rhomboid?

Change the information to:

A rhomboid does not have a right angle (or it is a rectangle)

If you can run the system now, does it perform correctly? Was it easy to change the information? Is there a way to look at the rules in a compact form?

4. INCOMPLETE INFORMATION

Enter:

A quadrilateral with 2 parallel sides is a trapezoid

As yet there is no information to classify a figure with no parallel sides. Is this discovered before or during run-time, or never? If never, what happens when you try to classify a figure with no parallel sides?

Enter:

A quadrilateral with no parallel sides is a trapezium

How does this change the system's performance?

5. GENERALIZATION

a. Discarding necessary attributes

Enter:

A trapezium with an interior angle of > 180 degrees is a concave trapezium

If you can run the system, does it correctly identify a trapezium without an interior angle > 180 degrees? Does the tool provide any way to check your information for completeness (to make sure the system does not give an answer for something you have not yet specified)?

Enter:

> **A trapezium without an interior angle of > 180 degrees is a convex trapezium**

You should now have all the information necessary to correctly classify any four-sided figure. If there is a way to look at the representation, is it compact? understandable? How does the system perform?

b. Discarding unnecessary attributes

In complex problem domains, the necessary attributes are not always known. The 5th attribute (whether all interior angles are equal) is unnecessary to solve the problem, but let us assume that fact is unknown.

Add the attribute for each of the 7 rules. Can you create an executable system? Has the new information changed or complicated the execution, or has that attribute been discarded as redundant? If you can look at a representation of the knowledge, has it changed?

6. DUPLICATING INFORMATION

Add one of the rules again. When does the tool inform you that you have already added the information?

7. CONCLUSION

While the classification of a four-sided figure would probably not be regarded as critical, some expert systems will make critical decisions. Does the tool give you any way to trace all the possible paths that can lead to a given conclusion?

D. Questionnaire

How would you rate the tool with respect to the following issues?

	Inferior	Poor	Fair	Good	Excellent	N/A
Ease of solving the problem with the tool						

Please elaborate:

Ease of acquiring new knowledge

Please elaborate:

Ease of refining knowledge

Please elaborate:

The understandability of the representation

Please elaborate:

Consistency checking
  of facts
  of the relations
   between facts

Please elaborate:

Insuring completeness (Covering the entire problem space, i.e. all possible combinations of attributes)

Please elaborate:

Efficiency/speed of execution.

	Inferior	Poor	Fair	Good	Excellent	N/A

Can the tool show you a concise representation of the information you put in? Could an expert readily understand it? Could an expert learn from it?

How long did it take you to solve the problem?

How effective was the benchmark in evaluating the feature? Can you think of a better problem or method for evaluation?

Background of evaluator.

Are you used to working with computers and software?
How familiar are you with AI products and concepts?
How well did you know the tool before trying to solve the problem?

```
/*
   Benchmark on aids to knowledge acquisition.
   Radian Corporation's solution to the problem of correctly classifying
   certain four-sided figures.
 */

MODULE: quad

DECLARATIONS:
[INTENT: "name\the 4-sided figure\"
 CHILD:   eq_sides
          par_side
          one_90
          gt_180                          ]

STATE: classify
  ACTIONS:
    rhombus              [advise "This figure is a RHOMBUS."]
    rhomboid             [advise "This figure is a RHOMBOID."]
    square               [advise "This figure is a SQUARE."]
    rectangle            [advise "This figure is a RECTANGLE."]
    trapezoid            [advise "This figure is a TRAPEZOID."]
    trapezium_cv         [advise "This figure is a CONCAVE TRAPEZIUM."]
    trapezium_cx         [advise "This figure is a CONVEX TRAPEZIUM."]

/*
    otherwise    [(advise "UNSPECIFIED COMBINATION OF PROPERTIES" , GOAL)]
    (used to ensure completeness in part 5)
 */

  CONDITIONS:
   all_sides_eq     [ eq_sides ]  { equal  not_equal }
   num_sides_par      [ par_side ]  { 0  2  4 }
   one_90               [ one_90 ]    { present  absent }
   gt_180                 [ gt_180 ]    { present  absent }

  EXAMPLES:
equal        4       absent        -            => (rhombus,GOAL)
not_equal    4       absent        -            => (rhomboid,GOAL)
equal        4       present       -            => (square,GOAL)
not_equal    4       present       -            => (rectangle,GOAL)
-            2       -             -            => (trapezoid,GOAL)
-            0       -             present      => (trapezium_cv,GOAL)
-            0       -             absent       => (trapezium_cx,GOAL)
```

```
MODULE: quad.eq_sides

DECLARATIONS:
[SILENT: "\the lengths of all 4 sides\are"
 OUT:      string result                  ]

STATE: only
  ACTIONS:
    yes  [ "equal" -> result ]
    no   [ "not_equal" -> result ]

  CONDITIONS:
    all_sides_eq
       [ ask "Are all 4 sides equal in length?" "yes no" ] { yes no }

  EXAMPLES:
   yes   => (yes, GOAL)
   no    => (no, GOAL)
```

```
MODULE: quad.par_side

DECLARATIONS:
[SILENT: "\the number of opposite sides parallel\"
 OUT:      string result                           ]

STATE: only
  ACTIONS:
    zero ["0" -> result]
    two  ["2" -> result]
    four ["4" -> result]

  CONDITIONS:
   num_sides_par
      [ask "How many sides have opposite sides which are parallel?" "0 2 4"]
                                                                  { 0  2  4 }
  EXAMPLES:
   0     => (zero, GOAL)
   2     => (two, GOAL)
   4     => (four, GOAL)
```

```
MODULE: quad.one_90

DECLARATIONS:
[SILENT: "determine if\a 90 degree angle\"
 OUT:     string result                    ]

STATE: only
  ACTIONS:
    yes  [ "present" -> result ]
    no   [ "absent" -> result ]

  CONDITIONS:
      one_90  [ ask "Is there a 90 degree angle?" "yes no" ] { yes no }

  EXAMPLES:
   yes  => (yes, GOAL)
   no   => (no, GOAL)




MODULE: quad.gt_180

DECLARATIONS:
[SILENT: "determine if\an interior angle > 180 degrees\"
 OUT:     string result                                  ]

STATE: only
  ACTIONS:
    yes  [ "present" -> result ]
    no   [ "absent" -> result ]

  CONDITIONS:
    gt_180
      [ ask "Is there an interior angle > 180 degrees?" "yes no" ] { yes no }

  EXAMPLES:
   yes  => (yes, GOAL)
   no   => (no, GOAL)
```

```
MODULE: quad
INTENT: "name\the 4-sided figure\"
 CHILD:   eq_sides
          par_side
          one_90
          gt_180

    STATE: classify
     IF ( par_side ) IS
        "0" : IF ( gt_180 ) IS
            "present" : ( advise "This figure is a CONCAVE TRAPEZIUM.", GOAL )
            ELSE ( advise "This figure is a CONVEX TRAPEZIUM.", GOAL )
        "2" : ( advise "This figure is a TRAPEZOID.", GOAL )
        ELSE IF ( eq_sides ) IS
            "equal" : IF ( one_90 ) IS
                "present" : ( advise "This figure is a SQUARE.", GOAL )
                ELSE ( advise "This figure is a RHOMBUS.", GOAL )
            ELSE IF ( one_90 ) IS
                "present" : ( advise "This figure is a RECTANGLE.", GOAL )
                ELSE ( advise "This figure is a RHOMBOID.", GOAL )

GOAL OF quad
```

(1st sample run)

```
How many sides have opposite sides which are parallel? [0 2 4] 4
Are all 4 sides equal in length? [yes no] y
Is there a 90 degree angle? [yes no] n

Advice: This figure is a RHOMBUS.

(RETURN continues) why


*****************************************************************

    Since a 90 degree angle is absent
        when the lengths of all 4 sides are equal
        and the number of opposite sides parallel is 4
        it is necessary to advise 'This figure is a RHOMBUS.'
        in order to name the 4-sided figure

        At <quad>
        c)ontinue e)laborate h)elp : c


*****************************************************************
```

```
(2nd sample run)

How many sides have opposite sides which are parallel? [0 2 4] 4
Are all 4 sides equal in length? [yes no] n
Is there a 90 degree angle? [yes no] y

Advice: This figure is a RECTANGLE.

(RETURN continues) why


*****************************************************************

    Since a 90 degree angle is present
        when the lengths of all 4 sides are not_equal
        and the number of opposite sides parallel is 4
        it is necessary to advise 'This figure is a RECTANGLE.'
        in order to name the 4-sided figure

        At <quad>
        c)ontinue e)laborate h)elp : c


*****************************************************************



(3rd sample run)

How many sides have opposite sides which are parallel? [0 2 4] 0
Is there an interior angle > 180 degrees? [yes no] y

Advice: This figure is a CONCAVE TRAPEZIUM.

(RETURN continues) why


*****************************************************************

    Since an interior angle > 180 degrees is present
        when the number of opposite sides parallel is 0
        it is necessary to advise 'This figure is a CONCAVE TRAPEZIUM.'
        in order to name the 4-sided figure

        At <quad>
        c)ontinue e)laborate h)elp : c


*****************************************************************
```

Appendix D

QUESTIONNAIRE RESULTS

SUMMARY OF QUESTION 2, QUESTIONNAIRE I

"Within the limits imposed by security or proprietary constraints, describe your ES task and its intended domain of application."

Commercial

- Trader's assistant for analysis in a commodities area
- Analysis of data from monitoring electricity usage in residential and commercial buildings
- Underwriting system for personal lines of insurance
- Network management, and banking and financial applications
- Financial advisor for businesses in automotive industry
- Monitor and analyze structure of public utility properties
- Diagnose events in high-voltage transmission lines
- Tax planning and securities trading

Computer Science

- Provide an environment for prototyping expert system applications
- Aid in design of hardware and software
- Generate and analyze system requirements specifications and high-level design model
- Convey knowledge gained about selecting expert system tools
- Configure computer components
- Develop a knowledge base for automatic test generation

Environmental

- Land and water management
- Multiple reservoir operation
- Hazardous waste management

Manufacturing

- Manufacture and testing of product
- Diagnose electronic malfunctions for B1-B aircraft avionics systems
- Test space shuttle main engine
- Develop tools for real-time and instrumentation use
- Implement a parts database to generate bills of materials

Medicine

- Diagnose arterial disease

Military Science

- Design and diagnose complex electro-mechanical systems
- Target identification from electronic sensor data
- Resource allocation in airborne radar sensor management
- Simultaneous examination of multiple courses of action
- Intelligent vehicle control for autonomous underwater vehicle
- Electromagnetic emission control system
- Electromagnetic signal exploitation system
- Reason about terrain maps
- Assist maintenance technician for B1-B aircraft

Space Technology

- Space vehicle system design: collection and organization of knowledge about missions, communication satellites, and antennas
- Command verification for free flying inspection robot
- Satellite diagnosis, fault isolation, and network control
- Configure remote satellite tracking stations

Teaching

- Instruct students in the theory and practice of using AI tools

SUMMARY OF QUESTION 3, QUESTIONNAIRE I

"What are the major issues and concerns involved in developing this application?"

The number preceding each category indicates how many people mentioned it as an issue or concern.

13 knowledge representation

- dealing with a constantly changing knowledge base
- flexibility
- developing knowledge manipulation tools to replace inference engine
- knowledge base structure
- model library of system templates
- discrete event simulation
- uncertainty handling
- viewing historic decisions
- developing a model-based expert system
- representing alternative plans/actions
- horizontal and vertical depth

9 delivery/fielding

- delivery vehicle
- maintenance/update
- establishing standards methods for validation
- expandability
- safety

9 domain/problem

- determining if there is sufficient human expertise
- forming a consensus among experts who disagree
- impact of expert system on domain
- who will train new experts?
- dwindling expertise
- monitor domain for shifts in environment
- developing project focus
- dealing with problem complexity
- distributed/cooperated problem solving
- teaching/learning AI techniques

8 integration

- with other hardware/software
- portability
- multiple rulebases

8 **control**

- dynamic replanning
- system autonomy
- concurrent/distributed systems
- concurrent/distributed users
- static/dynamic analysis

8 **user interface**

- different knowledge representation models for different users
- educating the user/maintainer
- how to display knowledge so it won't interfere with creativity
- explanation
- simplicity
- highly visual
- user acceptance of results

5 **knowledge acquisition**

- identifying flawed assumptions
- developing knowledge acquisition tools

4 **very large systems**

- knowledge bases
- databases
- rule bases
- multiple users

4 **performance**

- real-time
- computational speed

3 **reliability/consistency**

2 **cost/development time**

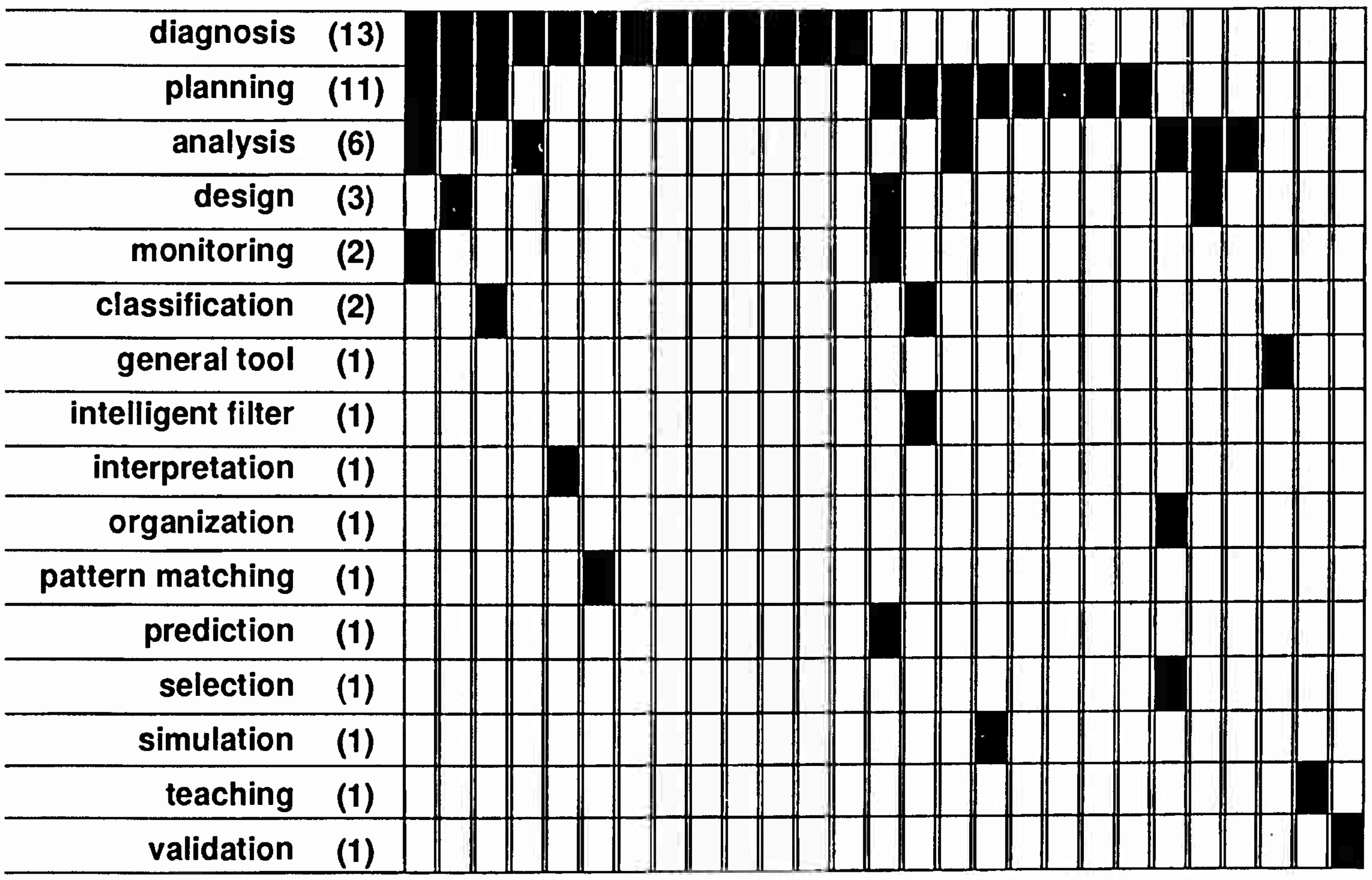
The following chart shows the distribution of the issues over people's responses. Each column represents one person's response.

knowledge representation	(13)	■	■	■	■	■	■	■	■	■	■	■	■	■												
delivery/fielding	(9)	■	■	■											■	■	■	■	■	■						
domain/problem	(9)	■			■	■	■	■							■						■	■	■			
integration	(8)	■			■				■	■						■	■							■	■	
control	(8)					■	■		■		■	■				■	■							■		
user interface	(8)				■	■	■		■	■	■		■												■	
knowledge acquisition	(5)				■	■		■		■					■											
very large systems	(4)	■			■	■			■																	
performance	(4)				■					■								■			■					
reliability	(3)							■		■									■							
cost/development time	(2)																		■							■

SUMMARY OF QUESTION 8, QUESTIONNAIRE I

How would you characterize your expert system task in terms of "standard" AI tasks such as diagnosis, planning, etc.?

Each column represents one person's response:



SUMMARY OF QUESTION 13, QUESTIONNAIRE I

What expert system tool(s) are you using?

Each column represents one person's response:

KEE	(9)	■	■	■	■	■	■	■	■	■																			
ART	(7)	■	■								■	■	■	■	■														
[in-house tool]	(6)										■	■				■	■	■	■										
RuleMaster	(4)																				■	■	■	■					
LISP	(3)			■												■									■				
M.1	(3)	■																		■						■			
Personal Consultant +	(3)	■																		■							■		
KnowledgeCraft	(2)		■																						■				
OPS-5	(2)															■	■												
Prolog	(2)																■											■	
S.1	(2)	■																		■									
ACORN	(1)										■																		
ADS	(1)	■																											
ESDE	(1)												■																
EXSYS	(1)	■																											
ExTran	(1)																												■
GURU	(1)	■																											
ROSS	(1)																	■											
YAPS	(1)																	■											

SUMMARY OF QUESTION 15, QUESTIONNAIRE I

"What evaluation criteria were used to select the tools?"

The number preceding each group of criteria represents how many used those criteria to select a tool.

13	representational structure/flexibility/expressiveness (rules, frames, schemes, semantic nets, methods, object oriented, syntax, uncertainty handling)
9	cost
9	speed/performance/efficiency/power/capacity
8	interface with end user (ability to customize, hide internals, graphics, natural language)
7	flexibility/modifiability/expandability
7	interface with developer (aids, documentation, on-line help, learnability)
6	inference/control mechanism (forward/backward chaining, blackboards, demons, active values)
6	support training available
5	portability/ability to upload to mainframe
5	underlying language available
4	ease of use
4	explanation capability/hypothetical reasoning
4	familiarity/liked by engineers
4	hardware/software required
4	integration capability (database, test data)
4	maturity/reliability/robustness of product/vendor
2	availability of tool in-house
2	truth/database maintenance
1	compiled/interpreted language
1	rapid prototyping ability

representation (11)	■	■	■	■	■	■	■	■	■	■	■												
cost (9)	■	■	■	■								■	■	■	■	■							
speed/power (7)	■				■	■						■	■				■	■					
flexibility (7)	■							■				■	■		■		■		■				
end-user interface (6)	■	■	■				■					■		■									
developer interface (6)	■	■		■				■		■										■			
vendor support (6)	■	■		■	■				■									■					
portability (5)	■				■		■		■				■										
underlying language (5)		■		■			■			■										■			
ease of use (5)	■												■			■				■	■		
inference/control (4)	■	■		■			■																
explanation (4)		■					■					■		■									
familiarity (4)	■														■				■			■	
hardware/software (4)		■	■										■									■	
integration (4)	■					■						■		■									
product/vendor maturity (2)				■									■										
in-house availability (1)																							■
truth maintenance (1)							■																
compiler/interpreter (1)													■										
rapid prototyping ability (1)			■																				

SUMMARY OF QUESTION 16, QUESTIONNAIRE I

"How long did it take to learn to use the ES tool(s) and what difficulties were encountered along the way?"

A few weeks: (six responses)

Learning syntax and the inference mechanism was not a problem, but teaching effective use was. Non-AI programmers tend to write procedural code (bad), and have a difficult transition period.

Knowledge engineers were previously trained; some differences were encountered with software version updates and run-time system.

Training is done over a week period with follow-on consultation as the project progresses; it is often difficult to articulate to the newer staff the rationale behind a particular approach.

Poor documentation and some system failure.

One month:
(six reponses)

Working through tutorials; no major problems.

One month after five-day training class; had trouble getting rules to fire, minor syntactic and naming conventions were confusing.

One month to proficiency. The programmer was familiar with other versions of Prolog; we were working with a beta release, so there were some bugs, and the documentation was incomplete/incorrect.

One month to become fluent and productive.

Preliminary system built five weeks after purchase of tool; interface was full screen, with multiple windows and interaction graphics. Knowledge was converted from a previous system.

Two months:
(four responses)

Six weeks; difficulties were hardware related.

Six weeks for non-programmer engineers; software bugs in new tools.

Two weeks to become a novice; six weeks to become knowledgeable; three months to become an expert.

Two months to feel comfortable.

A few months:
(three responses)

Initially a few months, but still learning.

Three months for a knowledge engineer.

Learning process took place over long period of time; learning depends on experience with similar tools; one should expect three or four months to go beyond superficial use.

Several months:
(five responses)

Months; bugs in hardware, lack of expert users to help.

Knowledge engineers were familiar with tool; two weeks training, several months experience, an eternity to learn to apply well.

Several months to become proficient; initial documentation was poor--now it is better.

Six months until comfortable; there were hardware/software problems; we didn't send anyone to the training course, which was a big mistake.

Still in the process.

SUMMARY OF QUESTION 17, QUESTIONNAIRE I

"How would you characterize the strengths and weaknesses of the expert systems tool(s) for your task?"

Strengths:

The number preceding each strength indicates how many people mentioned it as a strength.

6	utility as a prototyping tool / development environment
5	flexibility
5	powerful representation methods: rules/frames/schemes
3	well-designed/straightforward syntax
2	capacity
2	ease of use
2	good/simple inference mechanisms
2	portability
2	speed
2	support for user interface development
2	viewpoints
1	adaptability
1	allows for modular development
1	automatic inverse relations
1	availability of underlying language
1	built-in meta-information
1	can upload to mainframe
1	cardinality constraints
1	can compile to improve performance
1	documentation
1	efficiency
1	extensibility
1	hypothetical reasoning
1	integration
1	no Lisp required

1	no knowledge engineers required--normal engineers can enhance/maintain system
1	rule induction algorithm
1	truth maintenance
1	user-defined relations
1	value class checking
1	well-designed development tools

Weaknesses:

The number preceding each weakness indicates how many people mentioned it as a weakness.

4	unfriendly error messages/no support for error analysis
3	user interface simplistic/poor
3	speed/performance insufficient
3	representation not flexible enough
3	the more powerful a tool, the more difficult to use/inflexible it is
2	explanation not available/poor
2	inference methods weak/not flexible enough
1	blackboard capabilities poor
1	bugs
1	can't check consistency of rule class
1	can't make an autonomous system
1	can't test potential impact of new knowledge
1	cost of education
1	designed for small, specific tasks--do not support facilities/creative approaches
1	documentation confusing
1	graphics could be improved
1	inability to support features available
1	incompatibility

1	lacks a general belief system
1	less useful if logic is ill-defined
1	limited control of program execution flow
1	no access to external routines
1	no uncertainty handling
1	no user-defined relations
1	planning systems need a higher-level language for expressing goals and actions
1	tedious to obtain hardcopy rule output for documentation
1	too generic
1	worlds concept not developed

SUMMARY OF QUESTION 18, QUESTIONNAIRE I

"Describe your overall reaction to using expert system tool(s) based on your experience."

Most of the response was positive:

- "A very good tool, but hard to learn well. Tool is still evolving and needs a lot of functions."

- "We did not use a commercial tool."

- "It's nice to have a well equipped toolbox before you walk into the garage and open the hood."

- "Development of an expert system without a good tool package would be difficult. Only when an application has unusual constraints would I want to try this."

- "There is no perfect tool. The most appropriate tool must be selected for a specific job. Often the ability of the tool is dependent on the skill of the developer."

- "Enthusiastic."

- "Tools are useful as a quick approach, but because of their weaknesses, routines must be developed in Lisp/Prolog and then linked in."

- "Now that the tool's rule language is well integrated with the frame langauge, the tool could be more useful to us."

- "Expert system tools have allowed us to develop applications that could not be developed using traditional approaches; however, the tools are still primitive, and will not be mature for several years."

- "We are very pleased with the tool and the support."

- "We have just begun to appreciate the potential impact an expert system can have on our company. The development has taken longer than anticipated, but our overall reaction is very positive."

- "Tools need better ergonomics and intelligence."

- "No consistent development methodology. Good for rapid prototyping. Cannot handle large knowledge bases in a timely fashion."

- "Expert system tools are useful for rapid prototypes, but there are many cases where the tools do not meet system requirements (integration, performance, knowledge representation). Need an underlying language."

- "Expert system tools are useful for rapid prototyping, but there are many cases where these tools do not meet system requirements such as integration, speed, and knowledge representation."

- "Need access to another language."

- "No consistent development methodology. Good for rapid prototyping. Cannot handle large knowledge bases in a timely fashion."

- "Having been involved in the development of in-house tools, I feel the use of a commercial shell will provide a big advantage. Attention no longer not be focused on irrelevant software development."

- "Excellent for the problem at hand. It is very good for a domain expert to understand what an expert system is, what it can do for him, and how it can help in solving his problem. I/O and graphics are difficult to handle."

- "Very favorable for prototyping, especially for our limited time and staff."

- "I have yet to find a useful general purpose tool. Most are oriented to their prototyping application if not the domain itself. There is a lack of representational richness."

- "Positive; we plan to purchase different tools for different problems."

- "Extremely positive; the tool provides interfaces with other routines, and induction very nice for example-based inputs."

- "Better for programmer productivity--overall rapid prototyping. More difficult to integrate into overall application."

- "Tools are critical in the development of real knowledge systems. Delivery may require recoding, but it is worth it to derive the developmental power of any of the top expert system tools."

- "Although personal computer expert system development tools are still immature compared to tools that run on Lisp machines, within the next 1-3 years, the tools will overcome these weaknesses."

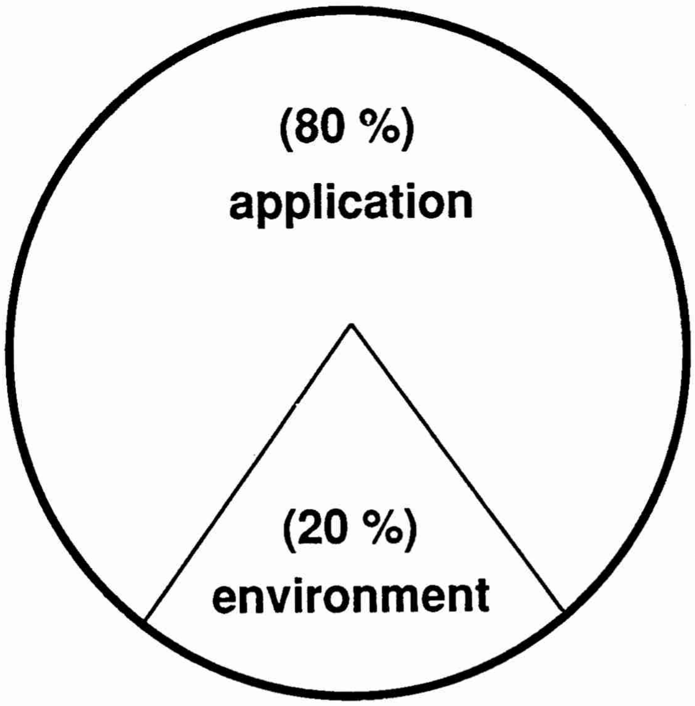
- "The tools are extremely valuable. In many cases the knowledge engineer doesn't have time to learn Lisp, but can learn a shell quickly. Very valuable for prototyping."

- "Better than no tools, but there is an urgent need for high level tools for specific classes of application."

SUMMARY OF EXPERIENCE QUESTION 2, QUESTIONNAIRE II

Are you building:

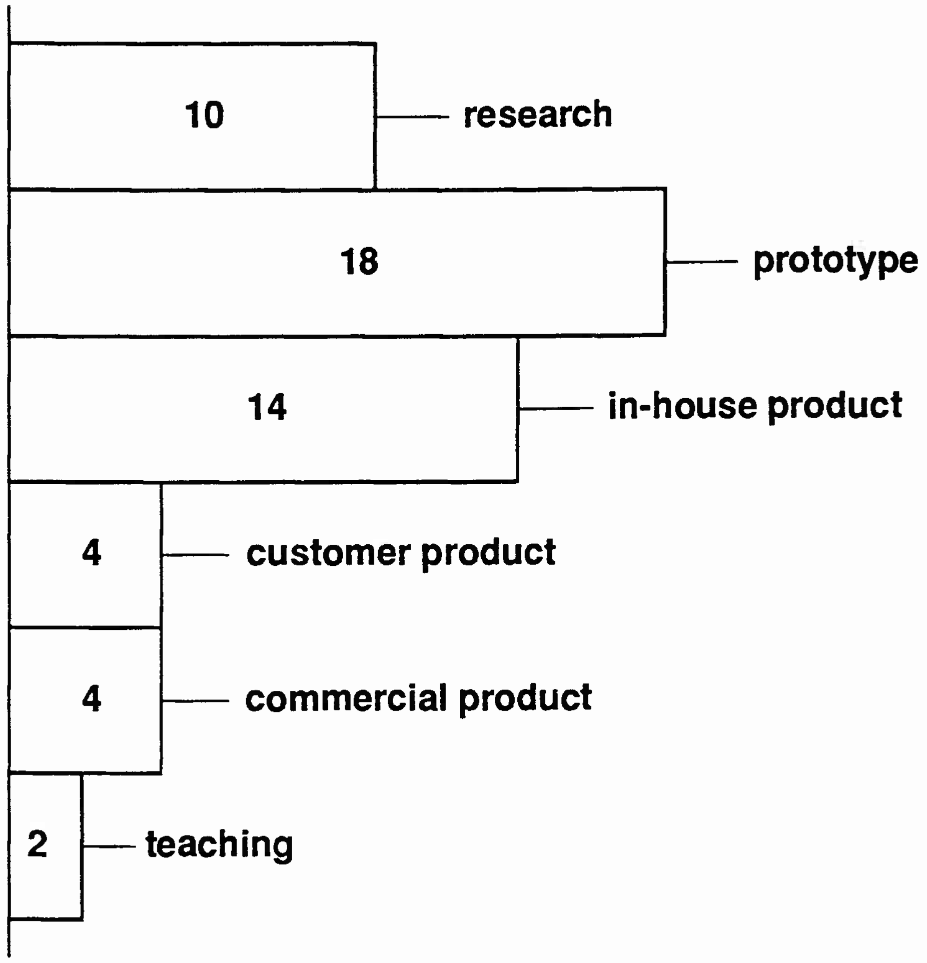
- an expert system application?
- an enhanced environment for building expert systems?



SUMMARY OF EXPERIENCE QUESTION 3, QUESTIONNAIRE II

Is the objective of your expert system task to:
(some responded in more than one category)

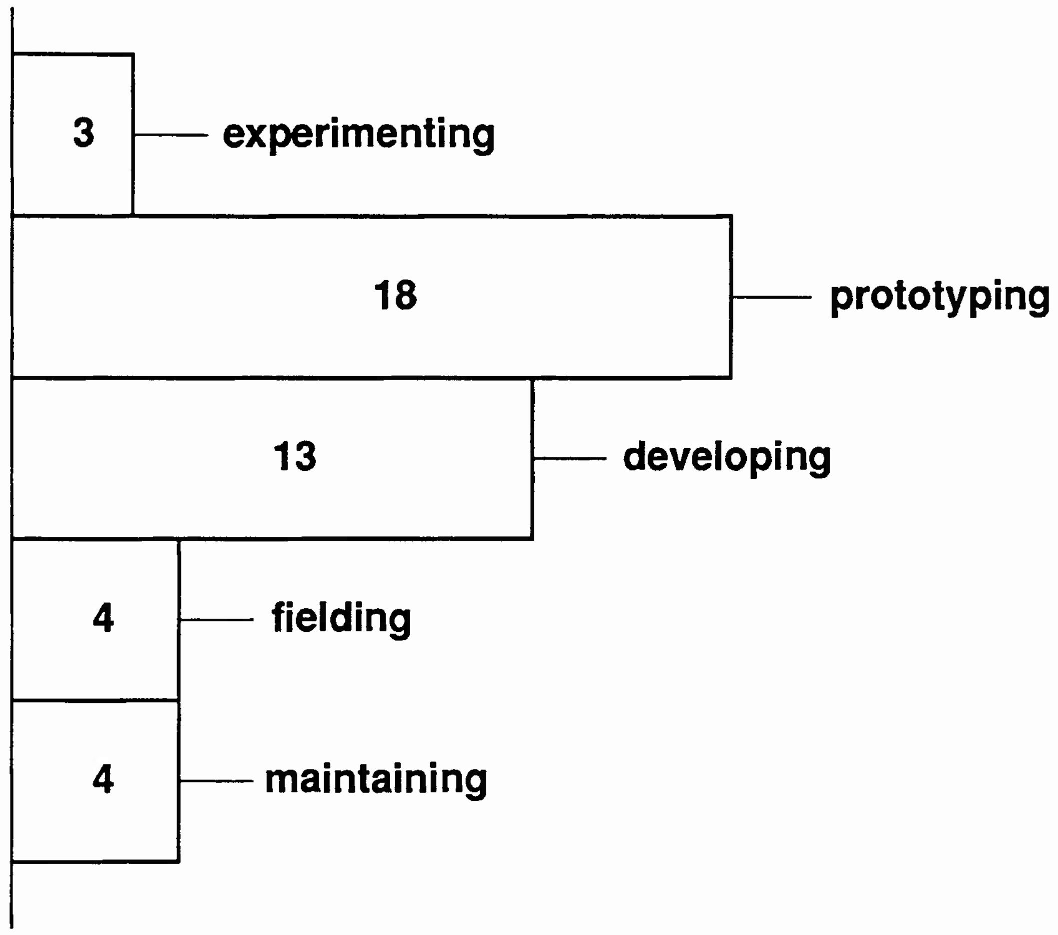
- do research?
- develop a prototype?
- develop an in-house product?
- develop a one-off product for a customer?
- develop a commercial product?
- teach people how to build an ES?



SUMMARY OF EXPERIENCE QUESTION 4, QUESTIONNAIRE II

Which stage of development best describes your expert system task at this time?

- experimenting/looking at tools
- prototyping/demonstrating feasibility
- developing
- fielding
- delivering/maintaining



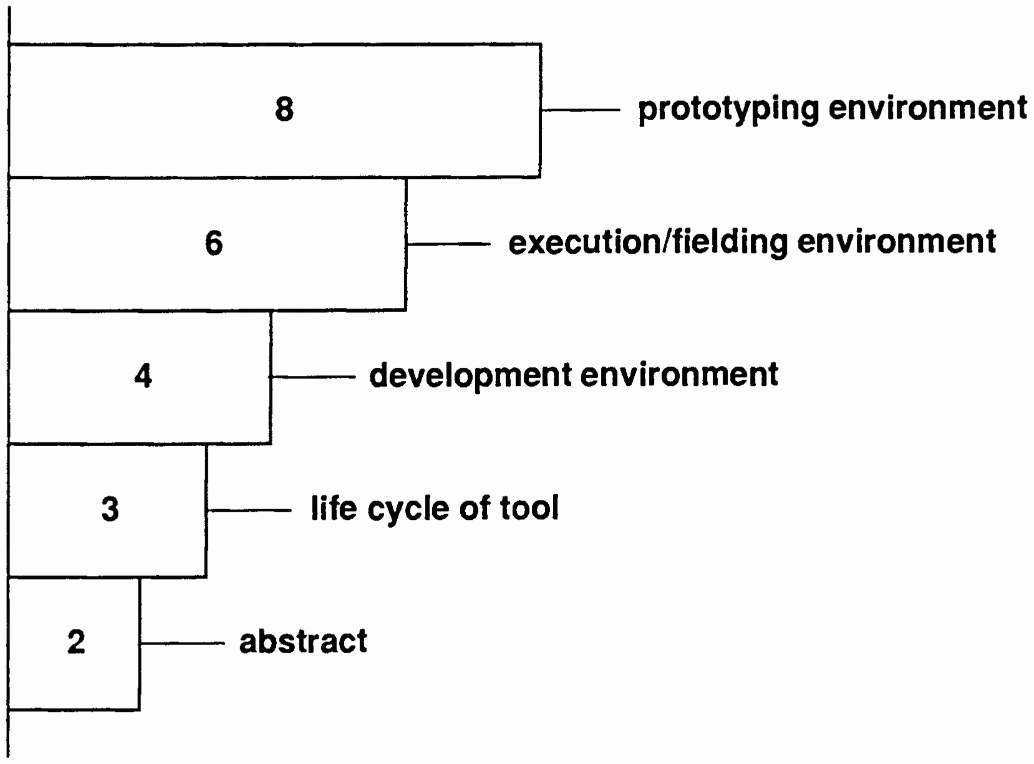
SUMMARY OF CONTEXTS FOR EVALUATION QUESTION 2, QUESTIONNAIRE II

These are some of the relevant contexts for evaluation:

- abstract
- prototyping environment
- development environment
- execution/fielding environment
- life cycle of the tool

Do you feel any of the contexts are more or less important than any others, and if so then why?

Here is a graph showing the responses. Each bar shows how many people thought that particular context was one of the most important.



SUMMARY OF PROBLEM CHARACTERIZATION QUESTION 1, QUESTIONNAIRE II

"Below are four ways of categorizing problems, and for each categorization, some problem types. Are there any other useful ways of categorizing problems?"

Here are the other categorizations that were proposed:

domain inference

The type of inferencing/reasoning required in the domain. Are decisions made from data (backward chaining) or from hypotheses (forward chaining)?

domain modelability

We can divide domains into those that can be readily modeled, and those that can't. Physical systems are readily modeled, such as electrical, chemical, math, and physics, whereas medical and financial areas are not easily modeled.

effect of the ES on organization

- influence on structural change
- influence on users
- resource commitment

expected end users

- domain expert
- lay person
- knowledge engineer
- computer professional

fielding requirements

- static/dynamic
- real-time
- probabilistic
- stochastic
- deterministic

maintenance team

- engineering experience
- number of people
- programming experience

system autonomy

Will the finished expert system be an autonomous system, or will it act as a consultant?

SUMMARY OF PROBLEM CHARACTERIZATION QUESTION 2, QUESTIONNAIRE II

"Below are four ways of categorizing problems, and for each categorization, some problem types. Would you add any types for any of the given categories?"

Problem domains:

chemistry
electronics
geological
information management
legal
manufacturing
mathematics
medical
military science
physics
space technology

Additions:

aerospace
business management
CAD
CAM
computer networking
engineering
finance
maintenance/repair
marketing/sales
resource management
risk management
software engineering
systems
telecommunications

Problem types:

classification
control
debugging
design
diagnosis
instruction
monitoring
planning
prediction
repair
simulation

Additions:

analysis
conceptual modeling/thinking
control
data fusion/reduction
data tracking
forecasting
intelligent access to
 information
integration
interfaces
interpretation
managing
prescription
scheduling

Complexity:

scope
size

Additions:

breadth
control required
depth
deterministic
ease of maintenance
epistomology/formal properties
isolated vs. cooperating with
   other expert systems
number of concurrent users
number of levels of structure
   chart
number of modules
probabilistic
rate of change of domain
   knowledge
readability
real-time requirements
representation difficulties
   of problem
static vs. dynamic
stochastic
volatility of knowledge base
well-understood

Development team characteristics:

AI experience
   knowledge engineers
   AI programmers
computer science background
domain experts
number of people
programming experience

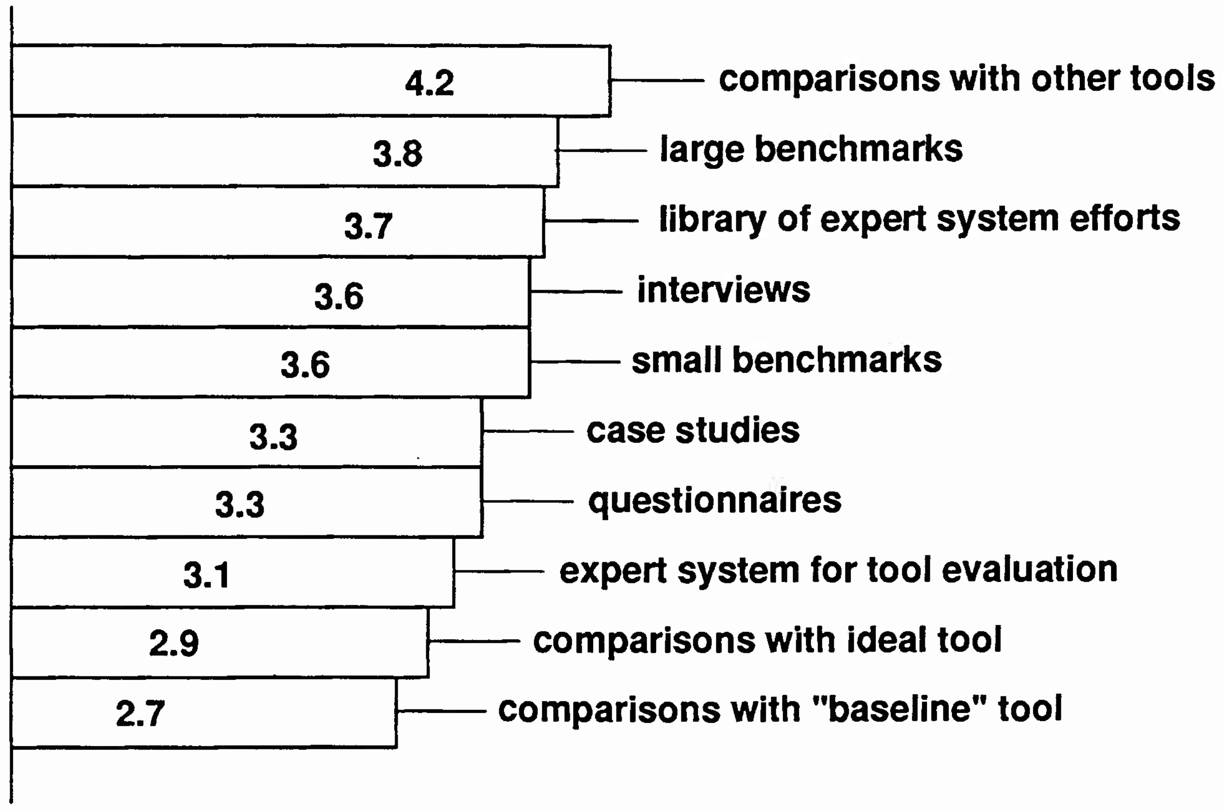
Additions:

creativity
knowledge acquisition skills
modeling expertise
other background (in
   linguistics, logic, math,
   physics, psychology,
   philosophy)
user interface experience
perception management (user,
   expert, management)

SUMMARY OF EVALUATION METHODS QUESTION 1, QUESTIONNAIRE II

For each evaluation method, please assign a usefulness rating from 1 to 5, 1 being "practically worthless" and 5 being "highly useful."

The ratings were averaged for each category, resulting in this graph:



RAND/N-2603-DARPA